České vysoké učení technické v Praze

Fakulta elektrotechnická

Bakalářská práce

Parametrizovaný generátor náhodných booleovských funkcí

Tomáš Měchura

Vedoucí práce: Ing. Petr Fišer

Studijní program: Informatika a výpočetní technika

leden 2006

Poděkování

Rád bych poděkoval panu Ing. Petru Fišerovi za ochotu a vůli při pomoci s vypracováním a dokončením této práce.

Prohlášení

Prohlašuji, že jsem svou bakalářskou 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 orávem autorským a o změně některých zákonů (autorský zákon)

V Praze dne 27.1.2005 .....

Abstract

This work describes the making of generator of boolean function in th form of PLA table. It tries to find effective solution of generating large number overlaping terms. The second part delas with suitable testing of boolean minimizer BOOM. The tests are done with many views on various parameters of minimalized function. From these tests it tries to make some conlcusions, which method of minimalization is better for partivular parameter.

Anotace

Tato práce pojednává o tvorbě generátoru náhodných booleovských funkcí formou PLA tabulky. Snaží se najít vhodná efektivní řešení při generování velkého počtu překrývajících se termů. Druhá část práce se zabývá vhodným testováním minimalizátoru BOOM-II. Testování probíhá z mnoha různých pohledů na parametry minimalizované funkce. Z těchto testů se pak snaží vyvodit závěry, která minimalizační metoda je lepší pro dané parametry booleovské funkce.

Obsah

Seznam obrázků

Seznam tabulek

Slovník pojmů

Literál - proměnná nebo její negace

Term - Term je vyjádřením součtu nebo součinu literálů. Pokud se jedná o součin, jedná se o součinový term, neboli P-term. Jedná-li se o součet, nazveme jej součtový term nebo-li S-term.

Minterm - součinový term, který obsahuje všechny vstupní proměnné. pro n vstupních proměnných existuje 2^n různých mintermů.

Maxterm - součtový term, který obsahuje všechny vstupní proměnné.

on-set - stav proměnných logické funkce, ve kterých je funkční hodnota rovna 1.

off-set - stav proměnných logické funkce, ve kterých je funkční hodnota rovna 0.

dc-set - stav proměnných logické funkce, na kterých daná fuknce nezávisí.

don't care - označení proměnné v PLA tabulce, které znamená, že daný term na této proměnné nezávisí.

Implikant logické funkce - Jedná se o výraz ve tvaru P-termu, pro který platí, že danou funkci implikuje; tzn. jestliže nabývá hodnoty logické 1, daná funkce nabývá též hodnoty logické 1. Implikant nazveme přímým implikantem právě tehdy, když po vypuštění libovolného literálu přestává být implikantem.

Iterativní minimalizace - minalizace založená na skutečnosti, že některá část minimalizačního procesu je řízena náhodou. Takže pokud algoritmus spustíme vícekrát na stejný problém nemusí vždy vést ke stejnému výsledku. Navíc, kombinací vygenerovaných implikantů z jednotlivých běhů iteračního procesu minimalizace, lze dosáhnou výsledků lepších

1 Úvod

Cílem této práce bylo vytvořit parametrizovaný generátor náhodných booleovských funkcí s výstupem formou PLA tabulky. A poté pomocí jím generovaných tabulek s různými parametry otestovat citlivost některých booleovských minimalizátorů. Nejprve jsem považoval celou úlohu za velmi jednoduchou, ale postupem času jak jsem se více a více dostával do nitra problému generování booleovských funkcí došel jsem k závěru, že vše není tak jednoduché jak se na první pohled zdá.

Na začátku mi p. Fišer poskytl jeho jednoduchý generátor, na kterém jsem se seznámil s vyskytujícími se problémy při generování. Tyto problémy jsem se snažil v této práci vyřešit, případně omezit různými metodami, což bylo na celé práci nejtěžší a nejvíce frustrující, poněvadž některé techniky byly nakonec silně kontraproduktivní. Pro můj generátor jsem se rozhodl zefektivnit a doplnit o potřebné funkce již zmíněný generátor p.Fišera, radši než začínat tvořit úplně nový. Avšak v průběhu implementování jsem díky novým vlastnostem musel podstatnou část hlavního algoritmu přepsat, takže to nakonec vyšlo nastejno.

Další část mé práce bylo ozkoušet vliv generovaných boolovských funkcí s různými parametry na výsledky vybraných minimalizátorů. Z čehož pak udělat jakýsi závěr, v jakých připadech je lépe použít jaký druh minimalizátoru. Poněvadž jsem s minimalizátory doposud neměl žádné zkušenosti. Tak mě tato část práce velmi lákala i přesto, že jsem si byl vědom hrozné otročí práce na přípravě testů a zpracovávání výsledků. Počátkem testovací fáze práce jsem se dozvěděl že p. Fišer je autor projektu BOOM-II a že by ho tímto způsobem rád otestoval.

1.1 Požadavky na generátor

Zadané požadavky na generátor se mi zdály poměrně snadné i když to byla menší část celé bakalářské práce.

- schopnost generovat PLA tabulky typu fd a typu fr.
- počet vstupů, výstupů a počet termů řádově do tisíce
- možnost nastavení poměru generování don’t cares, jedniček a nul na vstupu a výstupu
- možnost nastavení jisté granularity u generovaných don‘t cares, jedniček a nul jak na vstupu tak výstupu
- schopnost zpětně zjistit tyto poměry z již vygenerované PLA tabulky.
- dokumentace ke generátoru

1.2 Požadavky na testování

V testování byl kladen důraz na ucelenost testů a jejich správné vyhodnocení, především jednoznačně určit, který z minimalizátorů je na určitou PLA tabulku vhodnější.

- otestovat vygenerované PLA tabulky typu fr v dvou různých booleovských minimalizátorech. Zaměřit se především na různé parametry PLA tabulek
- vyzkoušet kombinace nastavení generovaných PLA tabulek, z toho určit zajímavé testy, které po té uskutečnit.
- výsledky testů vyhodnotit z co nejvíce různých hledisek

2 Generátor

Hned v úvodu práce mi p. Fišer nabídl pokračovat v jeho rozdělaném generátoru a poskytl mi zdrojové soubory. Zároveň mi upřesnil požadované parametry výsledného generátoru a hlavně mi vylíčil vyskytující se problém s překrývajícími se termy. Byl jsem v zásadě rád, poněvadž jsem měl na čem zakládat. Přesto jsem si uvědomoval, že některé doimplementovávané funkce to mohou udělat mnohem těžší, než kdyby se s nimi počítalo od začátku. V zadání bakalářské práce je zmiňována implementace vícehodnotové logiky. Po diskusi s p. Fišerem jsme se dopracovali k závěru, že to nebude potřeba, poněvadž každá vícehodnotová logika lze rozepsat na více binárních členů. Generátor měl generovat výstup formou PLA tabulky, typu fd a fr.

formát PLA

PLA tabulka se skládá z klíčových slov počínající tečkou a 1 alfanumerické matice kde každý řádek odpovídá jednomu termu a každý sloupeček odpovídá jedné vstupní nebo výstupní proměnné. Ve směru zleva do prava jsou nejdřív uvedeny všechny vstupní proměnné, až po té všechny výstupní. Symboly užité ve vstupní části matice:

- "0" značí že P-term obsahuje negovaný korespondující literál
- "1" značí že P-term obsahuje korespondující literál.
- "-" značí že P-term korespondující literál vůbec neobsahuje. tzv. don't care

Booleovská funkce je kompletně popsána trojicí on-set, off-set a dc-set. PLA tabulka používa čtyři typy různého zápisu funkce, uvedu zde pouze dva, které byly výstupem generátoru:

- typ fd - funkce je zadána množinou on-set a dc-set. Off-set je vypočitán jako doplněk jejich sjednocení.
- typ fr - funkce je zadána on-setem a off-setem. Dc-set je dopočítán jako doplněk jejich sjednocení.

Symboly užité ve výstupní části matice uvádím v tabulce 2.1:

	typ PLA	
	fd	fr
"1"	on-set	on-set
"0"	nemá význam	off-set
"-"	dc-set	nemá význam
"~"	nemá význam	nemá význam

Tabulka 2.1 význam symbolů ve výstupní části matice

Kompletní definice PLA formátu je k nalezení v [13].

2.1 Analýza problémů

Celý problém v generování booleovských funkcí byl pouze u výstupu PLA tabulky typu fr. Při náhodném vygenerování termu, který se s některým dříve vygenerovaným termem alespoň částečně překrýval, nesmělo dojít u kterékoli výstupní proměnné aby jeden term byl on-set a druhý off-set. Tato situace nastávala až při velkém procentu vygenerovaných mintermů obsažených v on-setu nebo off-setu oproti všem možným mintermům. Příklad:

Máme 10 vstupních proměnných, 2 výstupní, chceme generovat 1100 termů a vstupní procento don't cares je 0. To značí že budeme generovat přímo mintermy. Avšak počet všech možných mintermů je 2in kde in je počet vstupních proměnných, což je v tomto případě 1024, znamená to že minimálně dojde k 76 kolizím. Zde roste pravděpodobnost kolize lineárně.

Horší je to pokud máme jisté vstupní don't cares. Příklad:

Máme 10 vstupních proměnných, 2 výstupní, budeme generovat 300 termů a vstupní procento don't care je 20. Znamená to že budeme generovat termy kde nebudou obsaženy průměrně 20% * 10=2 literály. Takže tento term bude stát za čtyři mintermy. V případech se vstupními don't cares je pravděbodosnost kolize velmi obtížně vyjádřitelná. V nejlepším případě dojde zde ke kolizi po $2^{in(1-dc)}$ vygenerovaných termech.

Avšak pokud dojde ke kolizi neznamená to, že vygenerovaný term nepatří do výsledku. Pokud se ve všech výstupech se všemi s ním kolizními termy nevyskytuje jeden term v on-setu a druhý zároveň v off-setu pro stejnou výstupní proměnnou, tak je term vpořádku a je umístěn do výstupu. Při nulovém procentu výstupních don´t cares pravděpodbnost, že dva termy budou mít stejný výstup klesá exponenciálně s rostoucím počtem výstupů. Pokud zvýšíme don't cares na výstupu pravděpodobnost shody stoupá. Takže don't cares na výstupu začnou pomáhát při generování, až s počátkem kolizí generovaných termů.

2.2 Návrhy a řešení

2.2.1 Popis celého algoritmu.

Celý algortimus je velmi jednoduchý. Postupně se náhodně generuje term s danými pravděpodobnostmi don't cares a s daným poměrem jedniček a nul. Tyto pravděpodobnosti se pro každý term vypočítávají, aby se správně generovala nastavená granularita. Pokud generujeme tabulku typu fd tak se následující kontrola konzistence neprovádí a term se zapisuje normálně do výsledku.

V případě výstupu tabulky PLA typu fd se dělá test konzistence, aby ve funkci nebyl nějaký minterm který by byl zahrnut v on-setu a zároveň v off-setu. Po vygenerování náhodného termu, se hledají v hashtabulce možné kolizní termy, které se pak jeden po druhém porovnávají na překrytí s vygenerovaným termem. Pokud dojde k překrytí termu, tak se začnou porovnávat výstupní proměnné. Zde se kontroluje koznistence termu a to tím způsobem, že pokud jeden term má na výstupu v dané proměnné jedničku a druhý nulu, tak se prohlásí za nekonzistentní. Poté je přepsán jiným náhodným termem a znovu kontrolován s ostatními od začátku. Pokud je určen za konzistentní přidá se term do výsledku a zároveň se správně zařadí do hashtabulky. Pak se celý proces opakuje. Po úspěšném vygenerování počtu zadaných termů se výsledná tabulka uloží a pokud byl požadován výpis informace o PLA , spustí se algoritmus, který celý výsledek projde term po termu a na závěr vyhodnotí skutečná procenta don't cares, poměry 1:0, a metodou lieární regrese vypočítá pravděpodobnou granularitu těchto údajů. Pokud se ovšem zadaný počet termů nevytvoří do nastaveného času, tak se program ukončí.

2.2.2 Heuristika generování termů

Hned z počátku se nabízelo několik možností řešit problém kolizních termů, aby se zvedla pravděpodobnost úspěšného vygenerování konzistentního termu. Všechny se zakládaly na tom, že po několika po sobě jdoucích kolizích, se použije jedna z metod

- negenerovat vstupní proměnné náhodně, ale pokusit se vygenerovat s jistou náhodou nekolizní term nebo alespoň term s menším překryvem s ostatními termy
- vygenerovat náhodný term a zkusit najít nastavení výstupních proměnných tak, aby term mohl být zařazen do výsledku.

Po prodiskutovávání s vedoucím projektu p. Fišerem, byla druhá varianta zavrhnuta, poněvadž by vedla k jistému ovlivnění náhodnosti generované funkce. Proto jsem se zaměřil na první variantu. Tato varianta předpokládala, jistou datovou strukturu, ve které si budu

uchovávat nepoužité termy. Protože měl vyvýjený generátor generovat funkce s řádově tísícem vstupních proměnných, takováto datová struktura nebyla proveditelná, kvůli exponenciální závislosti počtu termů na počtu vstupních proměnných.

Další můj návrh byl použít jako datovou strukturu hash tabulku, kde hashovací funkce byla nastavitelný počet vstupních proměnných. V hash tabulce by se v jednotlivých položkách uchovával spojový seznam odpovídajících již vygenerovaných termů a počet termů v daném seznamu. V případě že by ve vstupní proměnné byl don't care tak by se předpokládaly obě možnosti a přidal by se term do hash tabulky vícekrát. Po rozboru této metody a zjištění že by při větších vstupních don't cares byla kontraproduktivní jsem metodu zavrhl. Heuristiku jsme totiž. potřebovali především pro funkce s velkým vstupním procentem don't cares. Poněvadž jsem další nápady na heuristiku neměl, soustředil jsem se na zefektivnění ostatních částí programu.

2.2.3 Zefektivnění hledání a vyhodnocování inkozistencí

Strukturu hash tabulky jsem nezavrhl a implementoval jsem jí do části programu, který hledá kolize. K danému termu najde v hash tabulce všechny adepty na kolizi a hledá jí tak v omezeném množství termů. Toto ovšem neplatí, stejně jako v předchozím případě, pro větší množství vstupních don't cares. Z předpokladu abychom neprocházeli větší množství termů s hash tabulkou než bez hash tabulky nám vychází nerovnost ze které určíme pro jaké procento don't cares by měla být tabulka teoreticky přínosná:

b - počet vstupních proměnných, které bere v úvahu hashovací funkce.

dc - číslo udávající poměr don't cares na vstupu

t - počet vygenerovaných termů´

Počet termů v celé hash tabulce ht je roven počet termů t krát průměrná dimenze termu tvořeného pouze z prvních b vstupních proměnných.

$$ht = 2^{dc*b} * t$$

Počet termů v jedné položce tabulky pt je počet všech termů v hashtabulce lomeno počtem ploţek

$$pt = \frac{ht}{2^b} = 2^{(dc-1)*b} * t$$

Počet termů ve kterých se bude hledat při použití tabulky je roven součinu pt a průměrnému počtu položek, ve kterých se bude kolize hledat. A ten musí být menší než počet termů. Z toho nám vyjde očekávaná hranice dc.

$$pt * 2^{dc*b} = 2^{(2dc-1)*b} * t < t$$

$$2^{(2dc-1)*b} < 1$$

$$(2dc-1) * b < 0$$

$$dc < 0{,}5$$

Výsledná teoretická hranice 50% vstupních don't cares nebyla v testech potvrzena, pravděpodobně díky větším nárokům při hledání v tabulce.

Hash tabulka byla implementována pomocí pole ukazatelů na jednotlivé položky. Každá položka byla spojový seznam ukazatelů na termy. Tím docházelo k šetření místa, poněvadž jsme tentýž term neukládaly vícekrát.

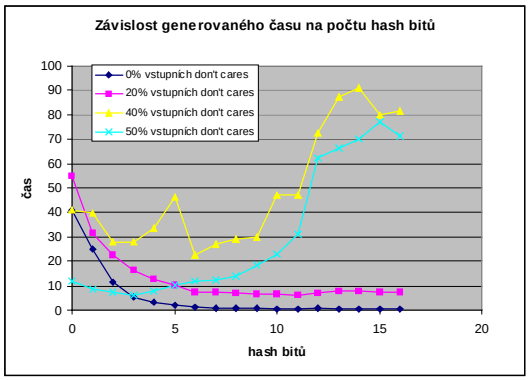
2.2.4 Ostatní vylepšení

Další zlepšení, které jsem implementoval, spočívá ve zkoušení vygenerování více kombinací výstupních proměnných pro jednu kolizní funkci. Toto přineslo nepatrné výsledky pouze v případech kde pomáhá i hash tabulka. Takže je v generátoru standartně nastavena na jedno opakování a není tato vlastnost dokumentovaná v příručce.

Po domluvě jsem do programu implementoval lineární granularitu. Původně jsem uvažoval i o dalších typech jako například kvadratická apod. Dále jsem ještě uvažoval o nějakém algoritmu který by průběžně kontroloval a zaručoval aby výstup měl opravdu požadované kvality především u vstupních don't cares. Návrh byl takový že algortimus bude průběžně vypočítávat zatím vygenerované statistiky a podle toho bude mírně pozměňovat vlastnosti dalšího generovaného termu, aby korigoval vznikající odchylky. Generátor však vykazoval chyby v generovaném input dc v průměru kolem 6ti procent, což jsem považoval za adekvátní. V těžších připadech s více output a vstupních don't cares se to zvyšovalo. Avšak v těchto případech by při použití zmiňovaného algoritmu na korekci chyby mohl celý generátor uváznout, protože by vyhovující již term nemusel existovat.

2.3 Rámcový test

Zde jsem testoval čas generovaní booleovské funkce v závislosti na počtu bitů hashovací funkce. Testy proběhly pro různá nastavení vstupních don't cares a to 0%,20%, 40%, 50% a 60%. Chci upozornit že pro každé nastavení don't cares byl jinak nastaven počet termů. Aby se daly křivky srovnávat v jednom grafu. Šlo mi především o porovnání průběhů křivky pro jiná don't cares a tím ověřit skutečnost, že při vyšších don't cares se bude účinnost generování zhoršovat. Podmínky tohoto testu jsou stejné jako byly v testech v odstavci 3.



Obrázek 2.3.1 Závislost času generování na hash bitech

Zde se ukázalo že teoreticky vypočítaná mez vstupního procenta don't cares byla poměrně přesná. Minimum času pro 0% don't cares bylo při použití 14ti hashbitů. Při 20% don't cares bylo minimum kolem 10ti hash bitů. Při 50% docházelo zlepšení pouze do 4 bitů.

Z toho vyplývá že lze úspěšně hash tabulku použít, ale musí se vědět kdy a v jakých případech. Při generování do 20% vstupních don't cares se může nastavit 10 bitů, při generování do 50% bych nechal nastavení do 5ti bitů. Pro větší procenta je již hash tabulka kontraproduktivní.

2.4 Závěr

Myslím si že se mi podařilo vytvořit užitečný nástroj na generování booleovských funkcí ve formě PLA tabulky, který je dostatečně výkonný a nabízí dost možností nastavení parametrů generované funkce. Program lze úspěšně využít pro generování booleovských funkcí do souboru testů vhodných pro testování booleovských minimalizátorů. Toto jsem úspěšně ověřil v druhé stěžejní části této práce.

3 Minimalizátory

Poté co jsem dokončil práci na generátoru, jsem se mohl začít plně věnovat další části bakalářské práce, která byla testování chování minimalizátorů. Jak již jsem několikrát zmínil, první co se muselo udělat, bylo vybrat alespoň dva různé booleovské minimalizátory, na kterých by proběhlo testování.

3.1 Analýza a výběr minimalizátorů

Na začátku práce mi p. Fišer vyvětlil, že si představuje abych jako minimalizátory použil jeho velmi dobrý BOOM-II. S tím jsem samozřejmě souhlasil. Přesto jsem se ovšem poohlídnul po jiných možných variantách, abych si zjistil co nejvíce o možných způsobech minimalizace a pochopil výhody a nevýhody zvolených minimalizátorů. V tom byl základ pro určení zajímavých testů a správného vyhodnocení výsledků.

Ze zjištěných informací jsem se utvrdil že BOOM-II je správná volba, poněvadž obsahuje 2 různé minimalizátory, které se navzájem doplňují. Kombinací obou dvou je vhodný pro velký počet vstůpních i výstupních proměnných. Ostatní minimalizátory se mi zdály mnohem méně univerzálnější nebo úplně nepoužitelné poněvadž například neumožňovaly vstup v PLA tabulce. Jmenujme jen některé z nich - ESPRESSO, Karnaugh Minimizer, Scherzo.

3.2 Popis funkce minimalizátorů

3.2.1 BOOM-II

BOOM-II je dvouúrovňový booleovský minimalizátor kombinující dva minimalizátory BOOM a FC-Min. Funkčně vychází z minimalizační metody Quine-McCluskey. Minimalizace probíhá obecně ve dvou fázích. První fáze je generování přímých implikantů vstupní funkce. Tato fáze probíhá iterativně a BOOM-II v ní volá podle nastaveného poměru jeden z minimalizátorů. Dochází tím k vyzužití výhod obou dvou minimalizátorů především při tvorbě skupinových přímých implikantů. Po každé iteraci se výsledky sloučí dohromady s předchozími. A dokud není splněno ukončující kritérium, tak program pouští další iterace. Jinak se přechází do další fáze.

Druhá fáze "problém pokrytí" je společná pro oba minimalizátory. Zde dochází k volbě minimální množiny přímých implikantů ze všech výsledků první fáze s ohledem na optimalizační kritérium. BOOM-II nabízí několik přístupů k řešení toho problému. Standartně je používaný algoritmus "Contributive Addition", který pro každý term počítá skóre a nejlépe ohodnocený term přidá do výsledku. To se opakuje dokud nepokryje celou funkci.

BOOM-II bere jako vstup PLA tabulku typu fr. Při spuštění minimalizace lze nastavit poměr v jakém mají oba minimalizátory běžet. Dále kritérium při kterém se program úspěšně ukončí (například po provedení N iterací, nebo uplynutí určitého času). Další důležité nastavení jsou optimalizační kritérium (literály, output cost, termy, atd.), jaký algoritmus bude řešit fázi "problém pokrytí" a v neposlední řadě výstupní formát. Poté se ještě dají volit různé nastavení heuristik a algoritmů specifické pro jednotlivé minimalizátory.

3.2.2 BOOM

BOOM = BOOlean Minimalizer

BOOM je minimalizátor vycházející z klasické Quine-McCluskeyho metody. Tato metoda minimalizace se uskutečňuje ve třech etapách.

V první etapě se nejprve vytvoří hyperkrychle dimenze počtu vstupů a poté se postupně zmenšuje její dimenze přidáváním literálů. Při tomto procesu se využívá heuristiky, která určuje který literál přidat, aby zredukovaná hyperkrychle pokrývala co možná nejvíce 1-termů.. Pokud již hyperkrychle nepokrývá žádný 0-term, pak je tato hyperkrychle implikantem funkce. Pokryté termy vyřadíme a začneme s novou hyperkrychlí. Takto

pokračujeme dokud nevyřadíme všechny vstupní on-set termy. Takto získané impikanty, nemusejí být přímé. Minimalizátor poté přechází do další etapy.

V této etapě "Implicant expansion" se minimalizátor pokouší z jich hotových implikantů odebrat literály, aby dosáhl přímých implikantů. V této fázi se dá využít více různých heuristik pro volbu literálů. Standartně se používá "Multiple sequential implicant expansion", při kterém se zkouší postupně odebírát všechny literály, a každý vytvořený přímý implikant se uloží. Poté se přejde na další implikant z předchozí etapy. Toto se opakuje do vyčerpání všech možností.

Pokud měla funkce více výstupů, opakujeme předchozí dvě etapy pro všechny výstupy zvlášť. Pak přejdeme do fáze "Implikant reduction", kde se ze všech přímých implikantů snaží vytvořit skupinové implikanty přidáváním literálů. Literály se přidávají sekvenčně až do doby, kdy je zřejmé, že daný implikant se nepoužije pro více výstupních funkcí. Tímto získáme pouze přímé implikanty.

3.2.3 FC-Min

Narozdíl od BOOMu a všech klasických minimalizátorů, FC-Min nevychází ze zadaných termů, ale začíná u výstupů, kde hledá shluky. Takže vlastně postupuje odzadu. Průběh algoritmu se dá rozdělit na tři etapy.

První etapa "Find Cover" vystihuje svůj název a dochází v ní k hledání shluků jedniček ve výstupu. Poněvadž je tento problém velmi složitý, dochází zde k jisté heuristice. Nejprve se vybere term který má na výstupech nejvíce nepokrytých jedniček a pokryje se co nejvíce z nich obdelníkem. Poté se přidá další term a snažíme se zvětšit obdelník aby pokrýval více jedniček. Počet takto přidávaných termů je voleno pravděpodobností při spuštění minimalizátoru. Pokud se již nepřidá term, označíme veškeré jedničky v obdelníku za pokryté a celý cyklus výběru termů opakujeme. Tato etapa se ukončí po pokrytí všech jedniček na výstupu.

Další etapa "Find Implicants" hledá k odpovídajícímu pokrytí implikanty. K danému pokrytí se tvoří implikant tak, že vezmeme všechny odpovídající termy k pokrytí a z nich utvořená minimální hyperkrychle je skupinový implikant. Tento způsob aplikujeme na všechna pokrytí postupně.

Další fáze "Input expansion" už se jen snaží rozšířit skupinové implikanty odebíráním literálů do oblastí s don't cares. Tímto dochází k lepším výsledkům minimalizace.aby minimalizovala output cost a jiné výstupní parametry. Pokud již nelze žádný term rozšířit, tak FC-Min končí

3.3 Závěr

Z výše popsaného fungování obou minimalizátorů a z testů uvedených v [13] jasně vyplývá, že by minimalizátor BOOM měl být účinnější na vstupy s mnoha vstupními proměnnými a málo výstupními. Naopak FC-min by měl být účinnější na vstupy s mnoha výstupy a menším počtem vstupů.

4 Testy

4.1 Úvod

4.1.1 Prostředí

V testování jsem použil PLA generátor verze 1.1 z ledna.2006 a BOOM-II verze 2.6 z listopadu 2005. Veškeré testy probíhaly různě na dvou strojích které byly velmi podobné, jak v hardwarovém vybavení tak v softwarovém vybavení. První počítač je o malinko pomalejší v rychlosti, avšak kompenzuje to o něco rychlejšími přístupovými časy do pamětí. V několika předběžných zde neuveřejněných testech jsem si ověřil, zda oba počítače dávájí stejné výsledky v několika různých odlišných nastaveních, aby jejich použití bylo relevantní a výsledky se daly navzájem porovnávat. Musím ještě podotknout, že test jako celek se vždy dělal pouze na jednom stroji. Zde je hrubá konfigurace obou z nich

PC1

Jedná se o procesor AMD Athlon XP+ bežící na frekvenci 1475 MHz, 512 MB RAM DDR 333 MHz, platforma Win2000 SP4.

PC2

Procesor AMD Athlon MP běžící na frekvenci 1575 MHz, 512 MB RAM DDR 333 MHz, platforma Win 2000 SP4.

4.1.2 Obecné nastavení

Veškeré testy se skládaly z více částí a byly vyhotovovány v pěti kopiích. Veškeré výsledky ze všech testů pak byly z těchto odpovídajících kopií z průměrovány a zde jsou udávany pouze tyto upravené výsledky. Toto opatření bylo děláno z důvodu ošetření možnosti statistické chyby.

První část byla generování vstupních PLA tabulek pomocí vytvořeného generátoru. Nastavení přepínačů generátoru bylo necháno na standartní nastavení a pokud se v testu nebyl proměnný vstup, výstup a počet termů tak byl nastaven na 100 vstupů, 20 výstupů, 100 termů, žádné don't cares, poměr jedniček a nul na 50:50, hash tabulka na 0 bitů. Tuto kombinaci vstupů, výstupů a termů jsem volil z předpokladu jistého obecnějšího použití. Předpokládal

jsem že většina funkcí, které jsou potřeba v praxi minimalizovat, má více vstupů než výstupů. Příkaz obecně vypadal takto.

generate_pr -i 100 -o 20 -t 100 -idc 0 -glidc 0 -odc 0 -glodc -idt 50 -glidt 50 -odt 50 -glodt 50 -h 0 in.XX.N.pla > in.XX.N.info

Kde XX bylo proměnlivé nastavení zkoumaných přepínačů a N bylo číselné označení kopie PLA tabulky.

Samotné testování minimalizátorů probíhalo ve všech testech ve třech nastaveních poměru běhu obou z nich a to 100% BOOM, 100% FC-Min a 50% BOOM s 50% FC-Min. Všechny testy pokud není jinak uvedeno probíhaly se základním nastavením všech přepínačů a to především nastavení ukončujícího kritéria na počet iterací 10 a optimalizačního kritéria na output cost+literals.

Nastavení ukončujícího kritéria standartně na 10 iterací bylo způsoben chybou v minimalizátoru BOOM-II. V průběhu minimalizace dochází k tzv. memory leak, který přibližně po 30ti až 50ti minutách začal vést k stále většímu swapování až nastala situace kdy program naprostou většinu času swapoval a již nic efektivně neprováděl. Tato situace se projevila skoro ve všech testech, kdy jsem nejprve musel metodou pokusu a omylu určit přibližné hranice problému, aby nedocházelo k onomu problému se swapováním. Musím podotknout že tento jev se projevoval u časově náročnějších testech, které byly pouze s minimalizátorem BOOM. FC-Min vykazoval řádově rychlejší zpracování výsledku. Příkaz vypadal přibližně takto

boom_pr -Si10 -FR 0 in.XX.N.pla out.0.XX.N.pla 2> out.0.XX.N.info

Kde XX bylo proměnlivé nastavení zkoumaných přepínačů a N bylo číselné označení kopie PLA tabulky. Význam přepínačů lze zjistit v manuálu k aplikaci [].

Vyhodnocování testů probíhalo z co nejvíce možných hledisek, co nám nabízel minimalizátor na výstupu. A to:

- CPU time - doba trvání výpočetního výkonu na dosažení výsledku
- output terms / defined terms - poměr termů po minimalizaci a termů před minimalizací.
- literals - počet lietrálů ve výstupní disjunktivní formě
- output cost - počet vstupů do všech výstupních OR hradel
- GEs - počet hradel potřebné k výrobě obvodu.

4.2 Test č.1 závislost počtu vstupů a počtu termů

4.2.1 Popis

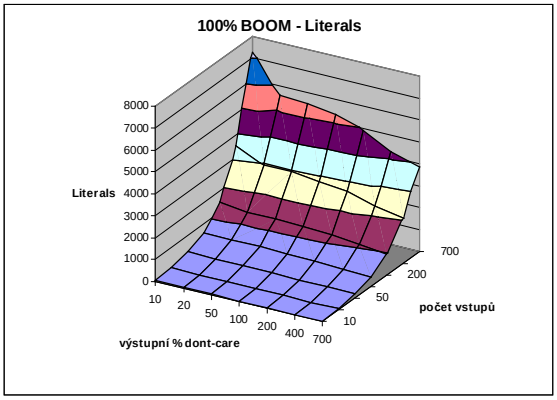
Tímto testem jsem zamýšlel otestovat jak se chovají oba minimalizátory při velkém počtu vstupů s nastaveným počtem výstupů na 20. Spíše to bylo postaveno na otestování FC-Minu při poměru počet vstupů/počet výstupů. Zároveň jsem byl zvědavý jak si poradí s velkým rozdílem počet vstupů a počet termů.

Trojrozměrné grafy uvedené v tomto testu jsou zkreslené. Poněvadž vynášené hodnoty na osách nereprezentují skutečnou vzdálenost od počátku osy. Je to způsobené omezením softwaru a je si třeba na to dát pozor, hlavně při určování lineárních a polynomických závislostí.

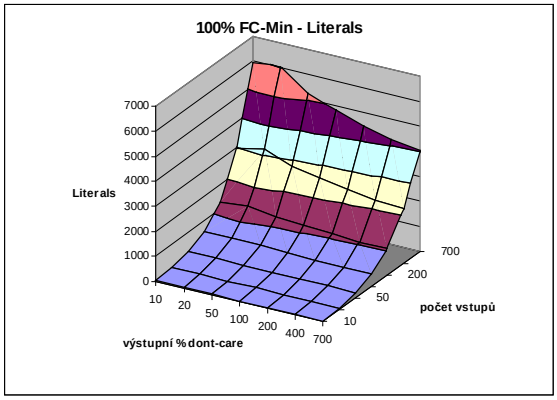
Rozsah počtu vstupních proměnných byl nastaven v sedmi krocích a to 10, 20, 50, 100, 200, 400, 700. Ve stejných krocích jsem volil i počet termů.

4.2.2 Výsledky

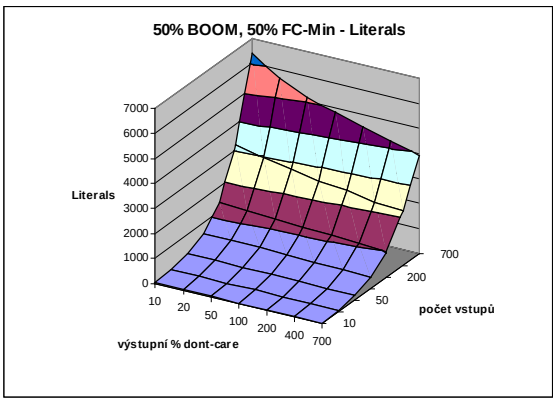
Literals



Obrázek 4.2.1 Test č.1 100% BOOM - Literals



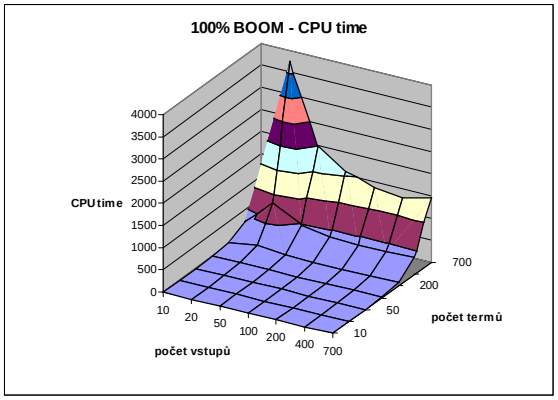
Obrázek 4.2.2 Test č.1 100% FC-Min - Literals



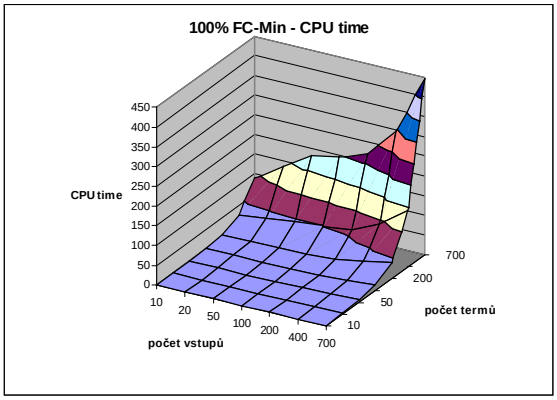
Obrázek 4.2.3 Test č.1 50% BOOM - Literals

V 10ti vstupních proměnných je na tom lépe FC-Min. Potom jsou na tom s BOOMem stejně. A až kolem 700 vstupních proměnných získává mírný náskok BOOM. Test jen pouze potvrdil předpoklady, že BOOM je lepší na mnohem více vstupů. Jen byla zajímavá hranice poměru vstupní, výstupní proměnné, kde se oba minimalizátory sobě vyrovnají.

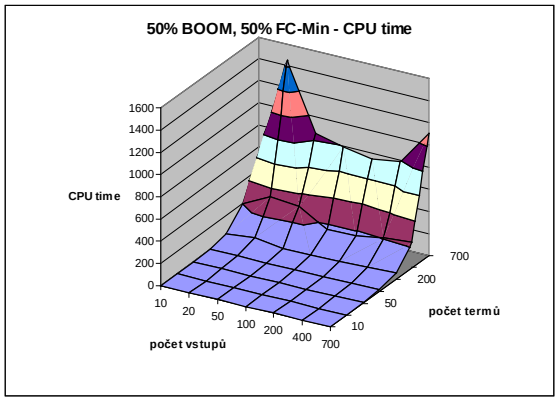
CPU time



Obrázek 4.2.4 Test č.1 100% BOOM - CPU time



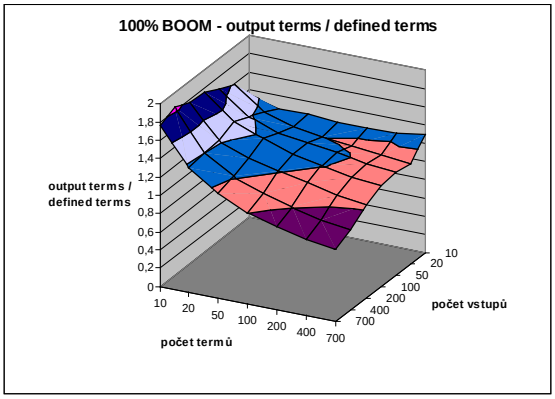
Obrázek 4.2.5 Test č.1 100% FC-Min - CPU time



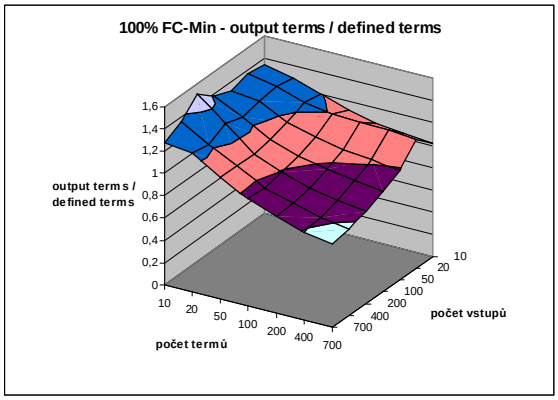
Obrázek 4.2.6 Test č.1 50% BOOM - CPU time

V čase je razantně lepší FC-Min. Při nízkém počtu termů se časy řádově rovnají avšak při počtu termů 700 již FC-Min vede až o jeden řád. Při 700 termech se také ukázala skutečnost kdy BOOM s rostoucím počtem vstupních proměnných zrychluje a FC-Min naopak. Je to samozřejmě způsobeno obráceným postupem obou dvou při hledání implikantů.

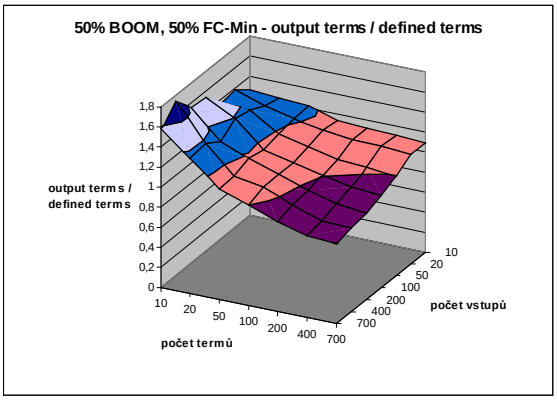
Output terms / defined terms



Obrázek 4.2.7 Test č.1 100% BOOM - terms



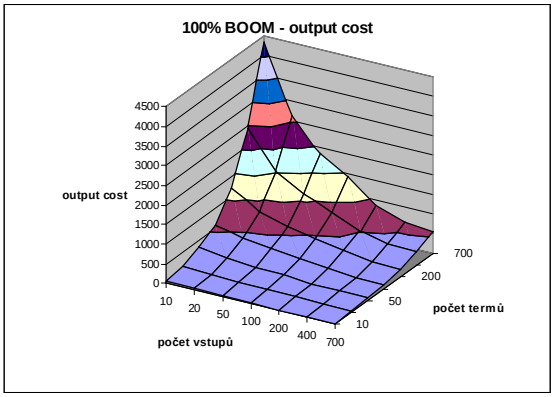
Obrázek 4.2.8 Test č.1 100% FC-Min - terms



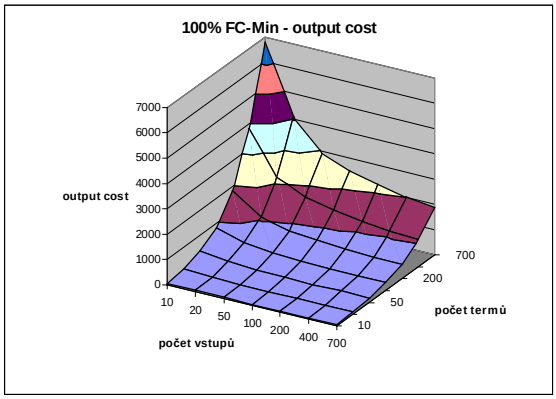
Obrázek 4.2.9 Test č.1 50% BOOM - terms

V počtu termů po minimalizaci vede ve všech případech FC-Min o pár procent před BOOMem. Avšak při počtu vstupů 20 dochází u FC-Minu k lokálnímu maximu. Předpokládám že to bude nějaká statistická chyba, protože nevidím důvod proč by tomu tak melo být.

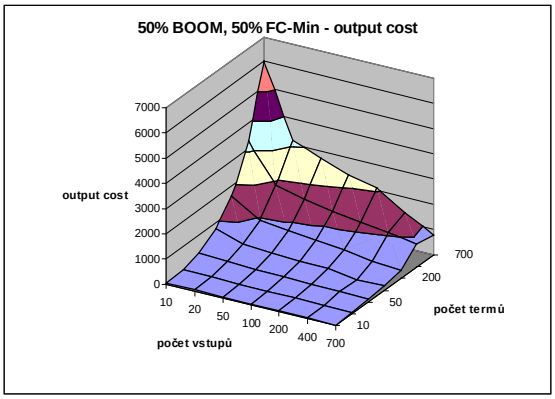
Output cost



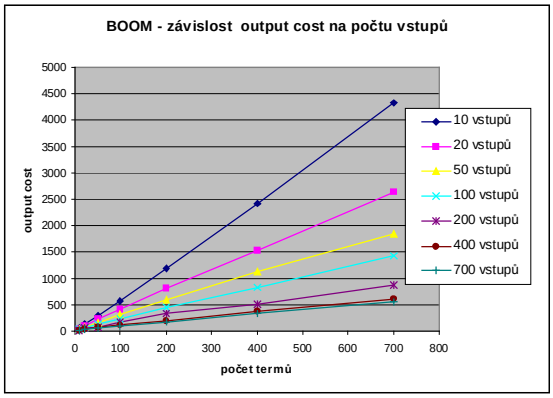
Obrázek 4.2.10 Test č.1 100% BOOM - output cost



Obrázek 4.2.11 Test č.1 100% FC-Min - output cost



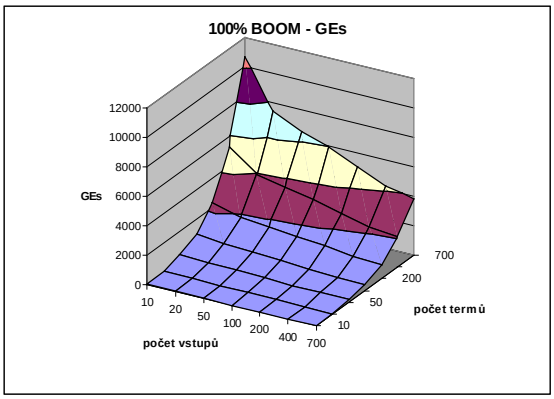
Obrázek 4.2.12 Test č.1 50% BOOM - output cost



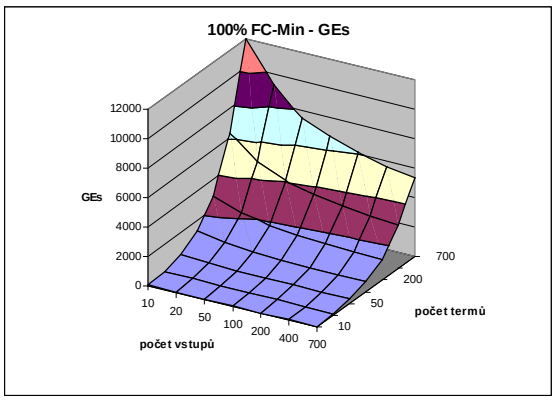
Obrázek 4.2.13 Test č.1 100% BOOM - output cost

Opět se zde potvrzuje že BOOM je výborný na velký počet vstupů a ve ve všech kombinacích termů a vstupů poráží FC-Min. V případě 700 termů a 700 vstupů až o více jak trojnásobek. Jak FC-Min tak BOOM vykazují lineární závislost output cost na počtu termů. Koeficient u lineární funkce je závislý na převrácené hodnotě počtu vstupů.

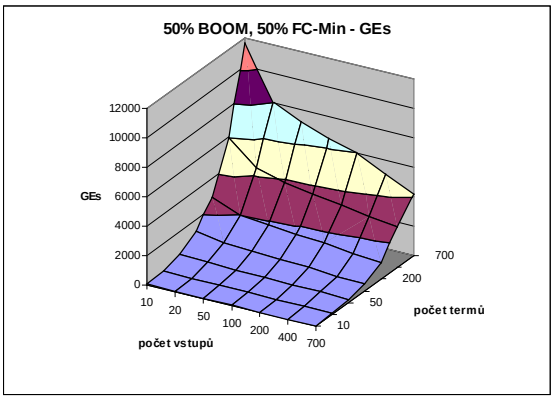
GEs



Obrázek 4.2.14 Test č.1 100% BOOM - GEs



Obrázek 4.2.15 Test č.1 100% FC-Min - GEs



Obrázek 4.2.16 Test č.1 50% BOOM - GEs

Také tyto výsledky nejsou ničím překvapivé. u obou minimalizátorů se projevuje téměř lineární závislost GEs na počtu termů. Ve velké míře dopadl lépe BOOM, avšak FC-Min zde podával srovnatelné výsledky.

4.2.3 Závěr

Tento test v zásadě potvrdil pouze to že na více vstupních proměnných je lepší použít minimalizátor BOOM. Při nastavení kombinace běhu BOOM a FC-Min dochází ke zprůměrování výsledků obou dvou minimalizátorů. Myslím si že při nastavení většího počtu iterací by mohl v tomto případě nést lepší výsledky.

4.3 Test č.2 závislost vstupního procenta don't cares a počtu iterací.

4.3.1 Popis

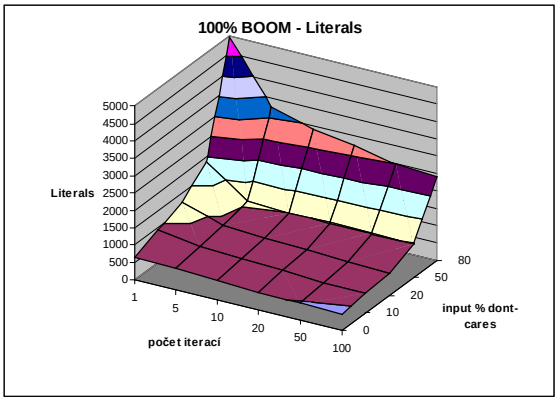
Tímto testem jsem chtěl ukázat že mnou standartně nastavený počet iterací na 10 je dostačující, že výsledná data nebudou výrazně nelineárně zkreslena od výsledků po 100 iteracích.

Trojrozměrné grafy uvedené v tomto testu jsou zkreslené. Poněvadž vynášené hodnoty na osách nereprezentují skutečnou vzdálenost od počátku osy. Je to způsobené omezením softwaru a je si třeba na to dát pozor, hlavně při určování polynomických závislostí.

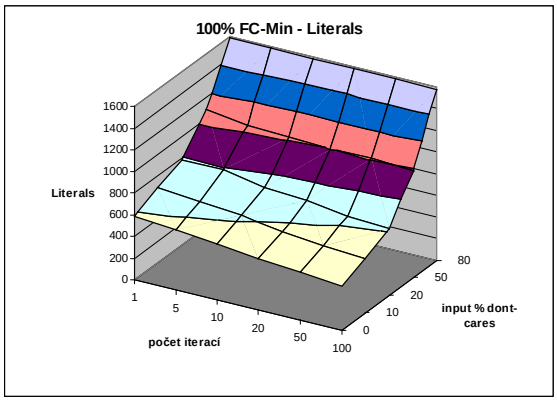
Počet iterací byl nastaven podle "pseudo" logaritmické řady na vzorky 1, 5, 10, 20, 50, 100. Druhá proměnná bylo vstupní procento don't care a to v tomto rozsahu 0, 10, 20, 50, 80. Počet vstupů byl nastaven na 100, výstupů 20, termů 100.

4.3.2 Výsledky

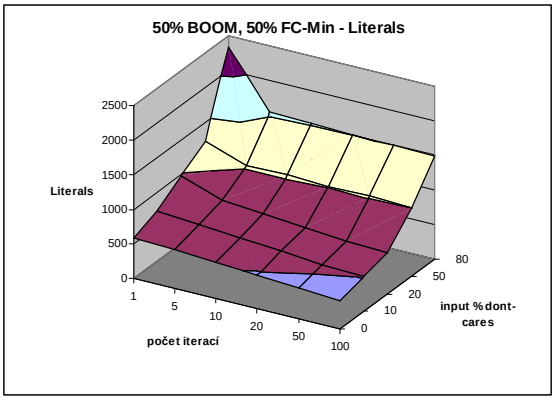
Literals



Obrázek 4.3.1 Test č.2 100% BOOM - Literals



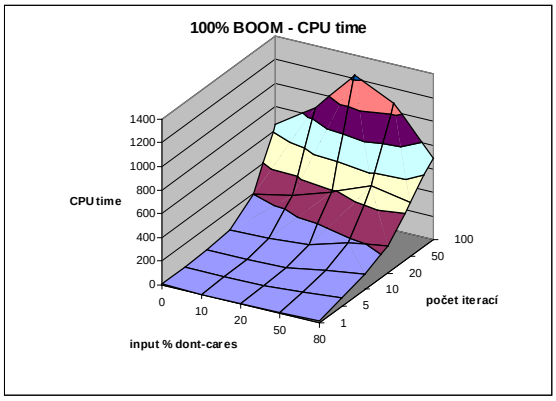
Obrázek 4.3.2 Test č.2 100% FC-Min - Literals



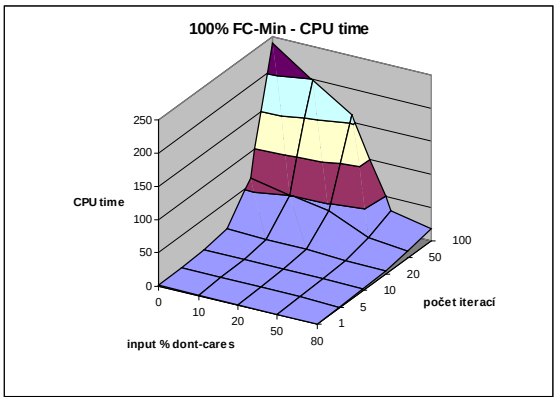
Obrázek 4.3.3 Test č.2 50% BOOM - Literals

V případě FC-Minu i BOOMu je závislost při nulovém procentu vstupních don't care v testovaném rozsahu téměř lineární. S rostoucím počtem iterací klesá počet literálů. Oba dva vykazují skoro tytéž výsledky. Avšak u FC-Minu se postupně závislost na iteracích snižuje s rostoucím počtem don't care na vstupu. při 80% již nedochází k ovlivnění výsledku počtem iterací. U BOOMu je tomu přesně naopak, původně téměř linearní pokles, se počtem vstupních procent don't care mění silně konvexní klesání literálů v závislosti na počtu iterací. V těchto vyšších procentech vstupních don't care je lepší minimalizátor FC-Min který je na tom lépe 2krát až 4krát. Zajímavé bylo že kombinace obou dvou vyšla skoro ve všech případech o trochu lépe než samotný FC-min.

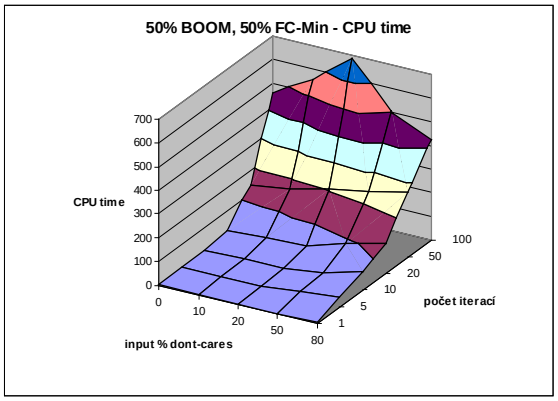
CPU time



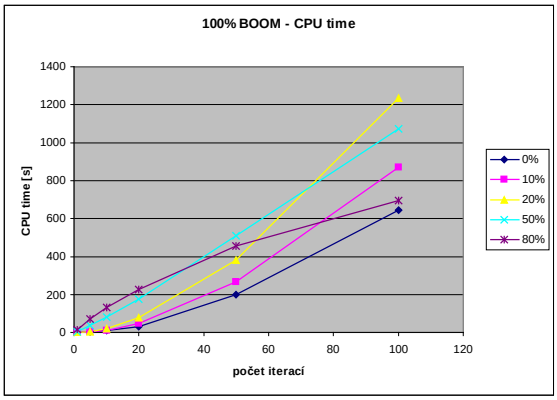
Obrázek 4.3.4 Test č.2 100% BOOM - CPU time



Obrázek 4.3.5 Test č.2 100% FC-Min - CPU time



Obrázek 4.3.6 Test č.2 50% BOOM - CPU time

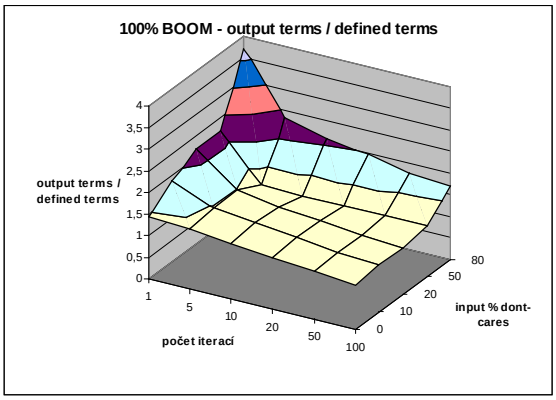


Obrázek 4.3.7 Test č.2 100% BOOM - CPU time

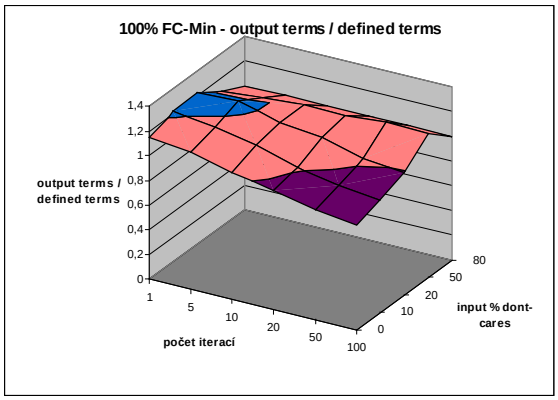
Celkově se dá říci že v čase byl lepší FC-Min. Zde ovšem byla výraznější tendence růstu času v závislosti na iteracích, nežli u BOOMu, který ovšem vykazoval mnohem horší časy. Zajímavé bylo pozorovat změnu obou minimalizátorů při rostoucím procentu vstupní don't care. Křivka závislost v BOOMu se zvyšovala ale v 50% se rostoucí konvexní funkce změnila na lineární a ve vyšších procentech dokonce na konkávní viz Obrázek 4.3.7 Kdyby tento trend pokračoval i pro více iterací, pak by to znamenalo že čas limtuje k nějaké hodnotě. Podle mne spíše dochází ke snížení počtu generovaných dříve nenalezených přímých implikantů v každé iteraci, takže by průběh měl mít dále lineární charakter.

FC-Min vykazoval zmenšování koeficientů u pravděpodobně polynomické křivky závislosti času na počtu iterací s rostoucím počtem procent vstupních don't care.

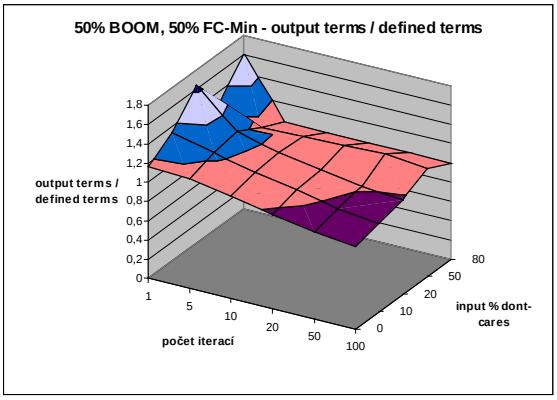
Output terms / defined terms



Obrázek 4.3.8 Test č.2 100% BOOM - terms



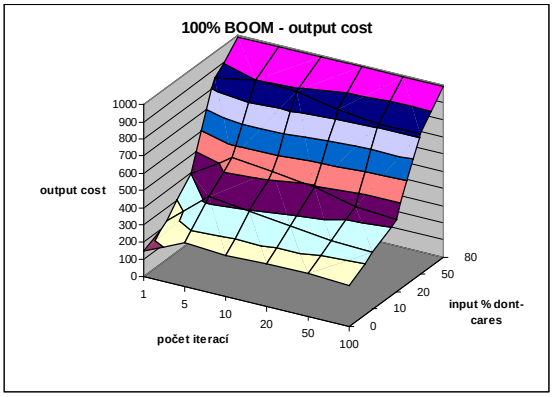
Obrázek 4.3.9 Test č.2 100% FC-Min - terms



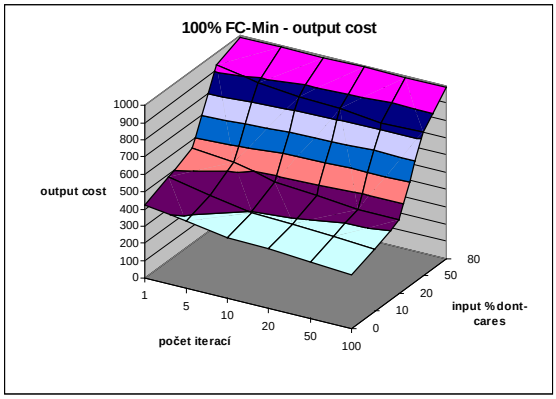
Obrázek 4.3.10 Test č.2 50% BOOM - terms

Zde se ukazuje pěkný rozdíl mezi BOOMem a FC-Minem. FC-Min již v první iteraci udělá relativně správný výsledek, který už jen po malých krůčkách zlepšuje. Zajímavé ještě je že v závislosti na procentu vstupních don't care vykazuje nejhorší výsledky kolem 50% poté se již zase zlepšuje. BOOM naopak v první iteraci udělá hodně špatný výsledek který dále razantně zlepšuje v nejbližších iteracích. Až výsledek po páté iteraci se dá relativně srovnávat s výsledkem po 100 iteracích. Také se výsledek zhoršuje se zvyšujícím se procentem vstupních don't care. Úspěšnost FC-Minu si vysvětluji lepším nalezením skupinových implikantů už při první iteraci.

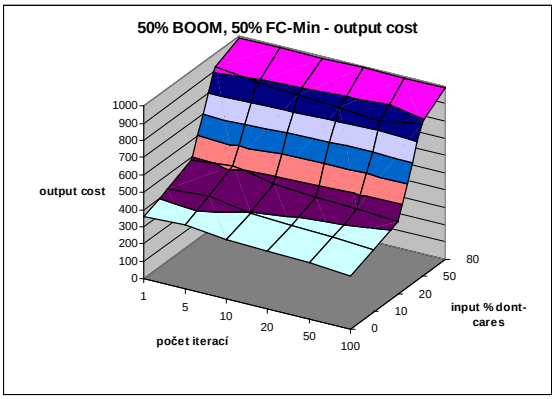
Output cost



Obrázek 4.3.11 Test č.2 100% BOOM - output cost



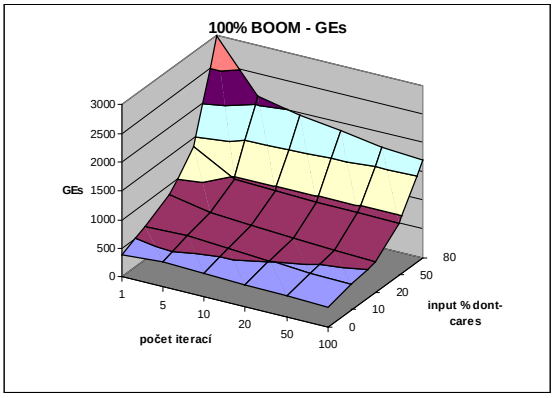
Obrázek 4.3.12 Test č.2 100% FC-Min - output cost



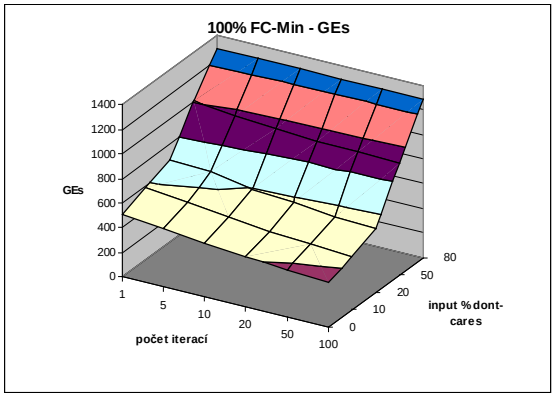
Obrázek 4.3.13 Test č.2 50% BOOM - output cost

Je zajímavé, že oba minimalizátory při 80% vstupních don't care měli totožné výsledky a nebyly ovlivněné počtem iterací. Což si vysvětluji tím že už v první iteraci našli oba dva nejlepší řešení, poněvadž bylo mnohem snažší hledat přímé implikanty. Celkově byl na tom lépe BOOM a dokonce první iterací udělal lepší výsledky než po 100 iteracích. Je to způsobeno pravděpodobně tím že je standartně nastaveno aby se minimalizovalo output cost + literals.

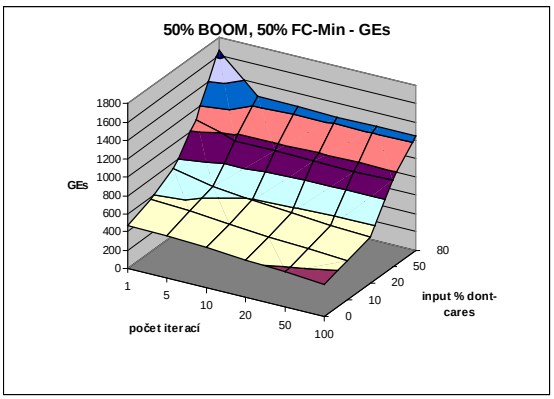
GEs



Obrázek 4.3.14 Test č.2 100% BOOM - GEs



Obrázek 4.3.15 Test č.2 100% FC-Min - GEs



Obrázek 4.3.16 Test č.2 50% BOOM - GEs

Zde byly oba minimalizátory na stejno při žádných vstupních don't cares. Při zvyšování si lépe vedl FC-Min, až se dostal na konstantní hodnotu nezávislou na iteracích při 80% vstupních don't care, která byla přibližně 2 krát lepší nežli u BOOMu. BOOM se choval se stejnými tendencemi jako u literálů. První iterací získal velmi špatný výsledek, který ovšem v pár nejbližších iteracích srazil na únosnou mez oproti 100 iteracím. Při spolupráci obou dvou došlo k průměrování výsledků

4.3.3 Závěr

V tomto testu se ukázalo že opravdu 10 iterací dává v drtivé většině relevantní výsledek oproti 100 iteracím. Během fáze z 10ti do 100 iterací se výsledek zlepší u BOOMu v průměru asi o 10%. FC-Min je ještě na počet iterací odolnější a v některých případech dává velmi slušné výsledky již v první iteraci. Časově je mnohem rychlejší za daných podmínek FC-Min. Kombinace obou dvou minimalizátorů jednou překvapila že dala výsledek lepší než samotné minimalizátory. Jinak dávala statisticky průměrné výsledky obou dvou.

4.4 Test č.3 závislost výstupního procenta don't care a počtu výstupů

4.4.1 Popis

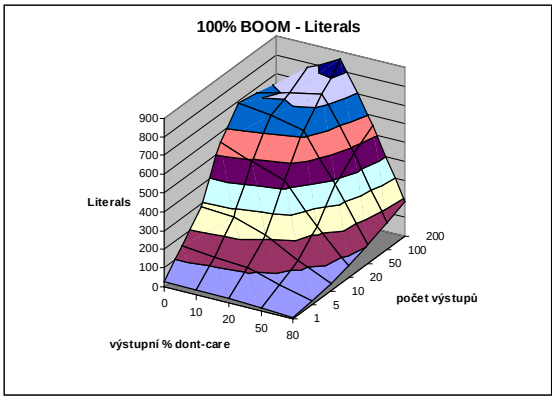
Tento test byl určen na testování závislostí při změně výstupů. Protože FC-Min začíná minimalizovat od výstupů, myslel jsem že by se mohly ukázat zajímavé skutečnosti.

Trojrozměrné grafy uvedené v tomto testu jsou zkreslené. Poněvadž vynášené hodnoty na osách nereprezentují skutečnou vzdálenost od počátku osy. Je to způsobené omezením softwaru a je si třeba na to dát pozor, hlavně při určování polynomických závislostí.

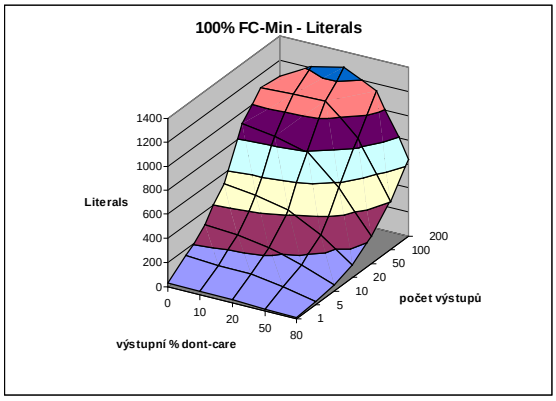
Počet výstupů byl nastaven na hodnoty 1, 5, 10, 20, 50, 100,200. Druhá proměnná bylo výstupní procento don't care a to v tomto rozsahu 0, 10, 20, 50, 80. Počet vstupů byl nastaven na 100, termů 100.

4.4.2 Výsledky

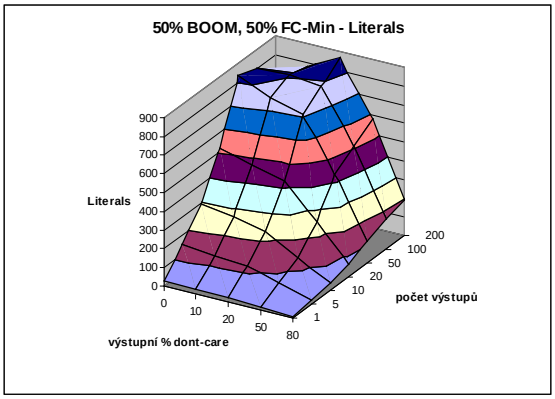
Literals



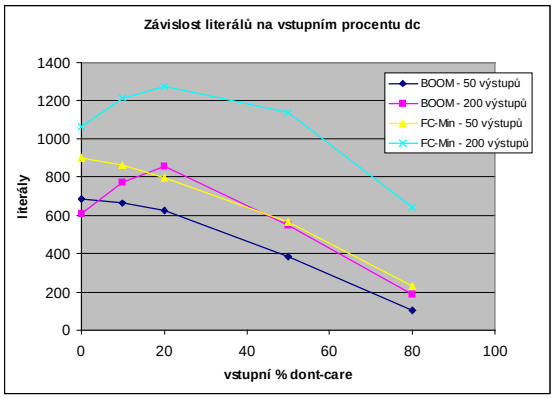
Obrázek 4.4.1 Test č.3 100% BOOM - Literals



Obrázek 4.4.2 Test č.3 100% FC-Min - Literals



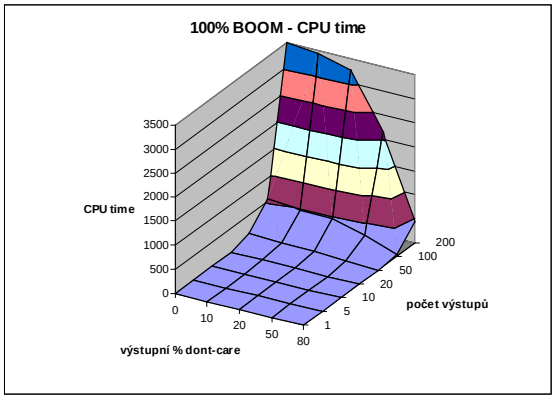
Obrázek 4.4.3 Test č.3 50% BOOM - Literals



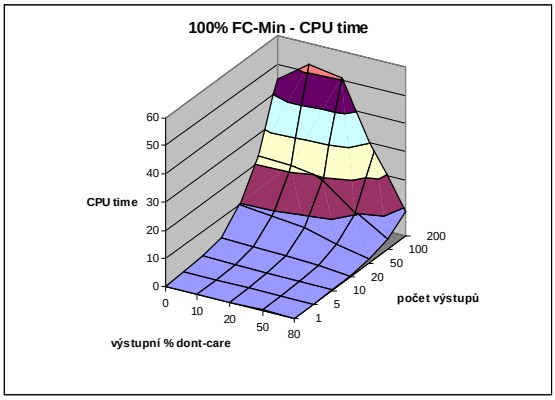
Obrázek 4.4.4 Test č.3 Srovnání - Literals

Zajímavé že nejvíce literálů je pro 20% výstupních don't care viz Obrázek 4.4.4. Tento jev si neumím rozumně vysvětlit. Možná je to způsobeno volbou malého počtu iterací, avšak při získávání dat v okolí tohoto jevu. tzn. Při 0%-20% input don't cares a 200 výstupů BOOM i FC-Min běžel poměrně dlouhou dobu a vznikalo již dříve zmíněné swapování. Pro 80% výstupních don't care literály ve zkoumaném rozsahu stoupaly lineárně spolu s počtem výstupů. V tomto případě byl na tom minimalizátor BOOM lépe ve všech situacích a to až o 80%. Kombinace obou minimalizátorů si vzala maximum z BOOMu poněvadž dopadla s ním téměř identicky.

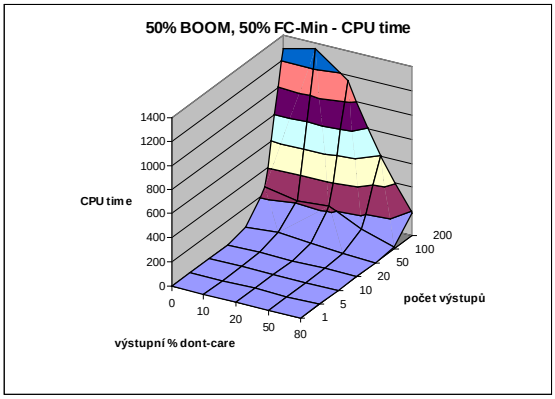
CPU time



Obrázek 4.4.5 Test č.3 100% BOOM - CPU time



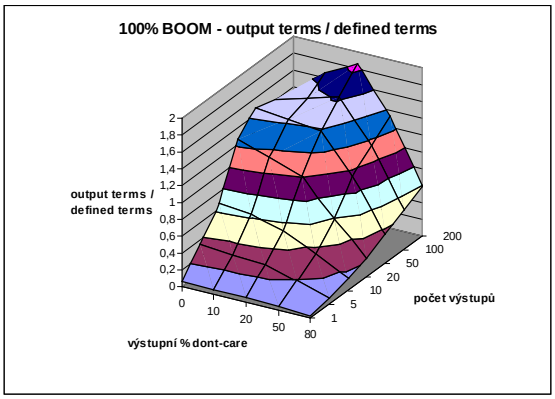
Obrázek 4.4.6 Test č.3 100% FC-Min - CPU time



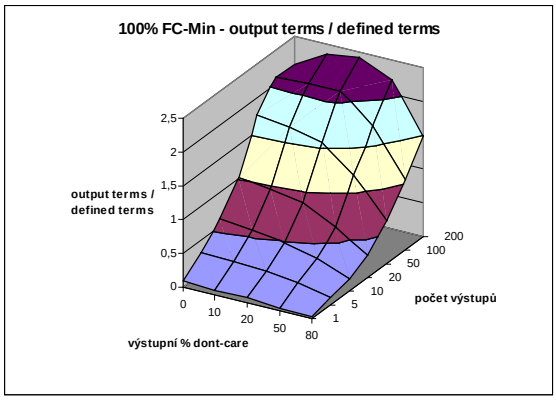
Obrázek 4.4.7 Test č.3 50% BOOM - CPU time

Zde se žádné překvapivé výsledky neodehraly. Podle očekávání časová náročnost u BOOMu rostla kvadratickou řadou v závislosti na počtu výstupech. Je to způsobeno vygenerováním velkého počtu implikantů, z kterých se potom sekvenčně snažíme rozšířit na skupinové. Časová složitost FC-Min rostla lineárně. BOOM byl jen o trochu rychlejší v při nastavení jednoho výstupu a občas pěti. Jinak v rychlosti kraluje FC-Min. Zvyšující se výstupní procento don't care jen snižovalo koeficient u křivek zavislosti na počtu výstupech. Bylo to pravděpodobně způsobeno tvorbou větších přímých implikantů a tím i snížení jejich počtu a menší časovou náročností.

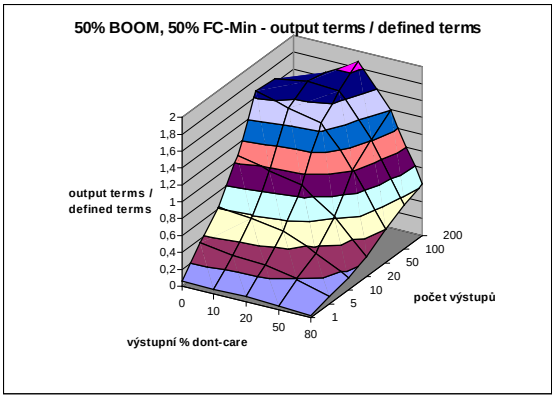
Output terms / defined terms



Obrázek 4.4.8 Test č.3 100% BOOM - terms



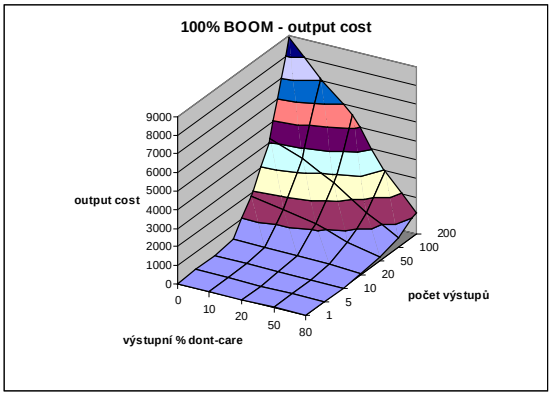
Obrázek 4.4.9 Test č.3 100% FC-Min - terms



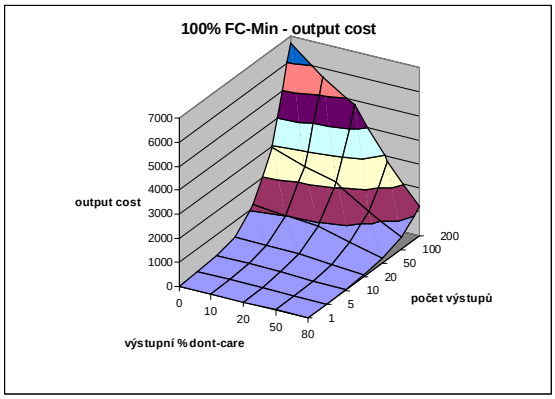
Obrázek 4.4.10 Test č.3 50% BOOM - terms

Tyto grafy jako by z oka vypadly kritériu literálů. Nedokážu si vysvětlit čím to je že, protože v jiných testech vycházely literály a termy rozdílně. Rozhodně platí to samé co jsem již psal výše. Lépe dopadl minimalizátor BOOM a v některých situacích je lepší než FC-Min až o 80%.

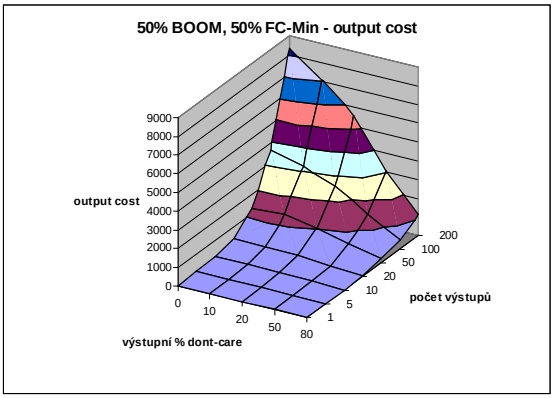
Output cost



Obrázek 4.4.11 Test č.3 100% BOOM - output cost



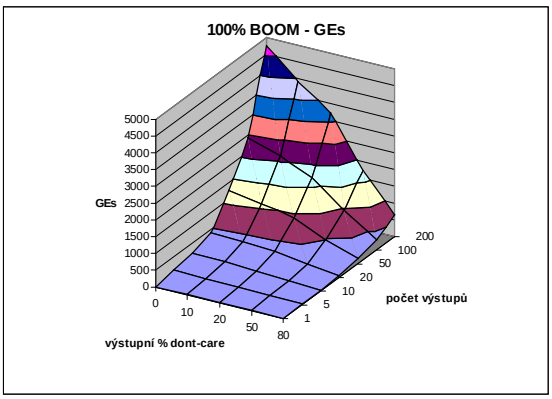
Obrázek 4.4.12 Test č.3 100% FC-Min - output cost



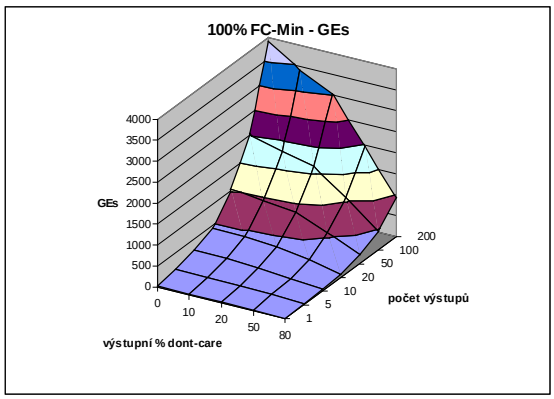
Obrázek 4.4.13 Test č.3 50% BOOM - output cost

V případech přibližně do 20 výstupních termů dává lepší výsledky BOOM. V ostaních případech je tomu naopak, takže kvalitnější výsledky dává FC-Min. Což není vůbec překvapující. Kombinace obou minimalizatérů, funguje spíše jako BOOM, ale jsou zde místa kde naopak kopíruje FC-Min. Dá se říct, že spíše se přiklání vždy k horší variantě. Takže bych kombinaci nedoporučoval použít.

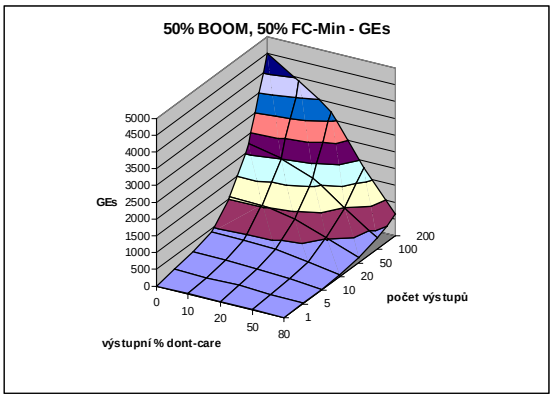
GEs



Obrázek 4.4.14 Test č.3 100% BOOM - GEs



Obrázek 4.4.15 Test č.3 100% FC-Min - GEs



Obrázek 4.4.16 Test č.3 50% BOOM - GEs

V Případě GEs lze říci že pokud je výstupní procento don't care nad 50% dává BOOM lepší výsledky nežli FC-Min. Je to pravděpodobně způsobeno tím že přímé implikanty genrované BOOMem lze pa díky výstupním don't cares jednodušeji rozšířit na skupinové. FC-Min nalezne velké pokrytí výstupu pomocí don't cares, ke kterým však pak h§ře nachází skupinové implikanty. Kombinace se spíše přiklonila na stranu BOOMu, je to logické, protože vygeneroval pravděpodobně kvalitnější skupinové implikanty.

4.4.3 Závěr

Tento test opravdu nepřinesl nic nového. Předpokládal jsem že počet výstupů v kontextu s procentem výstupních don't care bude mít výraznější vliv na FC-Min avšak ukázalo se, že ve velké míře byl lepší minimalizatér BOOM mimo kritéria output cost a GEs, kde bych doporučil při vyšším počtu výstupu než 20 použít FC-Min. FC-Min je obecně také mnohem časově méně náročnější protože má přibližně linearní časovou složitost v závislosti na počtu výstupů.

4.5 Test č.4 závislost vstupního poměru 1:0 a výstupního poměru 1:0

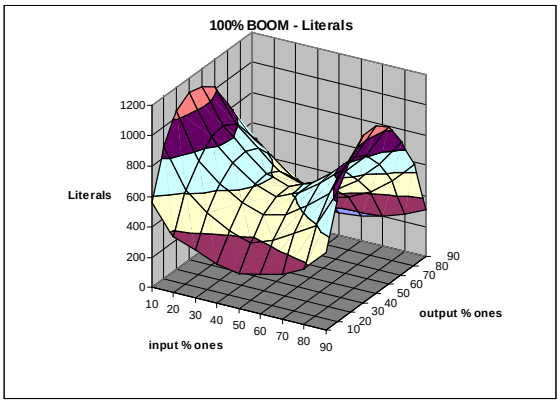
4.5.1 Popis

Tento test byl ze všech asi nejzajímavější poněvadž testoval něco nového, co ještě nebylo minimalizátory testováno. Tento test byl především určen na porovnání chování obou minimalizátorů v závisosti na různém poměru jedniček a nul v termech jak ve vstupech tak i ve výstupech.

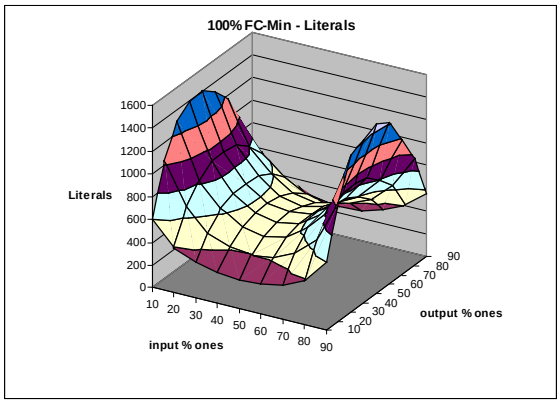
Rozsah vstupního poměru byl od 10% do 90% v korku po 10%, rozsah výstupního poměru byl taktéž od 10% do 90% v kroku po 10%. pocet vstunpích proměnných byl nastaven na 100, výstupních na 20 a počet termů na 100. Jako ve všech testech i tento byl test vyhodnocován na všechny výstupní kritéria.

4.5.2 Výsledky

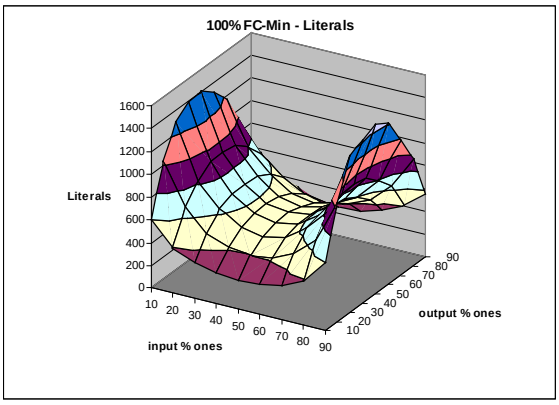
Literals



Obrázek 4.5.1 Test č.4 100% BOOM - Literals



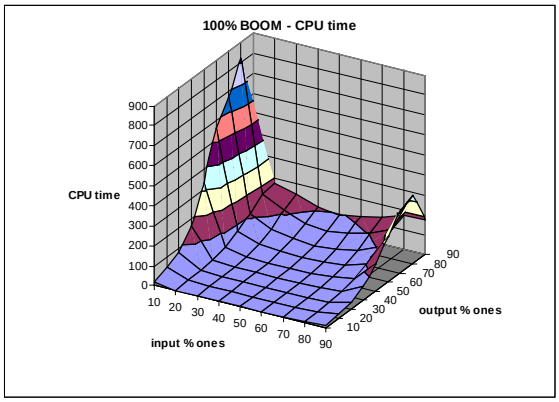
Obrázek 4.5.2 Test č.4 100% FC-Min - Literals



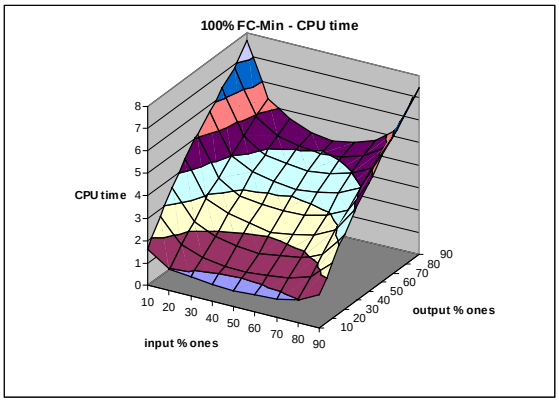
Obrázek 4.5.3 Test č.4 50% BOOM - Literals

Z grafů je patrné že z hlediska literálů v případech vstupů s velkým nepoměrem nul a jedniček byl lepší minimalizátor BOOM a to v případě vyváženého výstupu o až 1,5krát. Dále je z výsledků zajímavé že BOOM byl v případě většího poměru jedniček na výstupu (60+%) lepší než v případě malého počtu jedniček na výstupu. Nabízí se zlepšení celého algoritmu, tím že při zjištění většího počtu jedniček než nul na výstupu. V celém výstupu prohodíme jedničky za nuly a naopak. V případě přibližně stejného počtu nul a jedniček byl z hlediska litterálů o pár procent lepší algoritmus FC-Min.

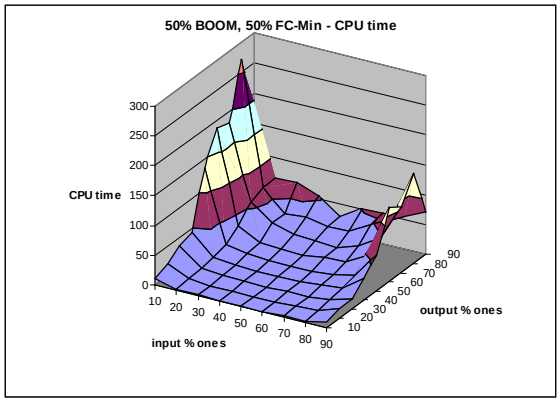
CPU time



Obrázek 4.5.4 Test č.4 100% BOOM - CPU time



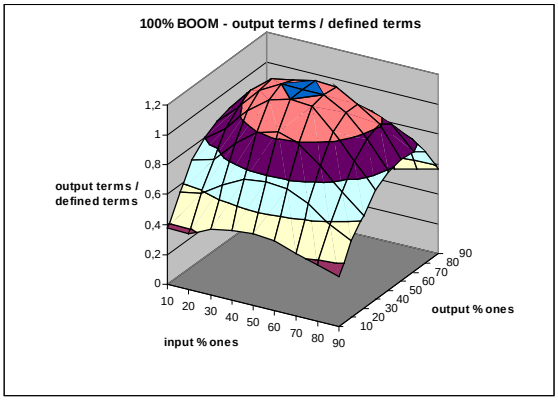
Obrázek 4.5.5 Test č.4 100% FC-Min - CPU time



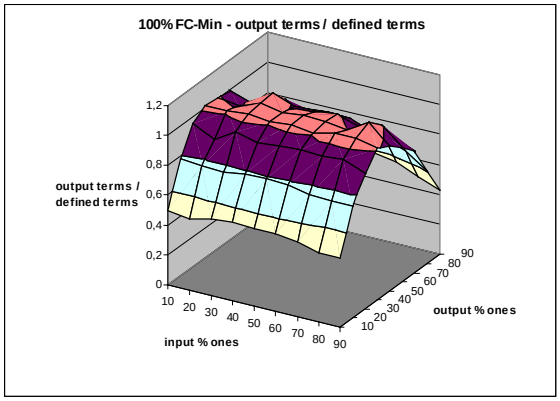
Obrázek 4.5.6 Test č.4 50% BOOM - CPU time

Ohledně spotřebovaného času na minimalizaci jasně vede minimalizátor FC-Min. V nejmenším rozdílu je asi 2x rychlejší a to v rozvnoměrném rozložení jedniček na vstupu a co nejméně jedniček na výstupu. V krajních mezích FC-Min vítězil až o 2,5 řádu. V nízkém počtu jedniček na vstupu i výstupu navíc BOOM vykazoval velký rozptyl času. tzn. velmi záleželo na vygenerované funkci.

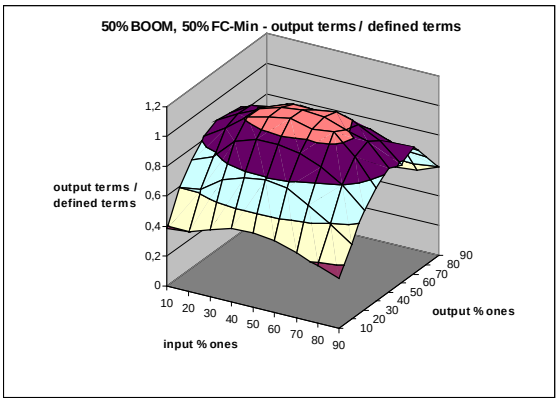
Output terms / defined terms



Obrázek 4.5.7 Test č.4 100% BOOM - terms



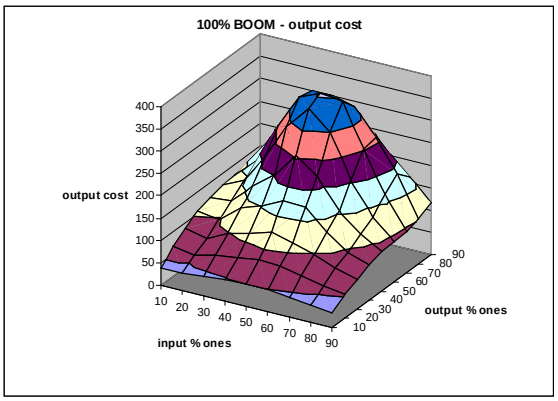
Obrázek 4.5.8 Test č.4 100% FC-Min - terms



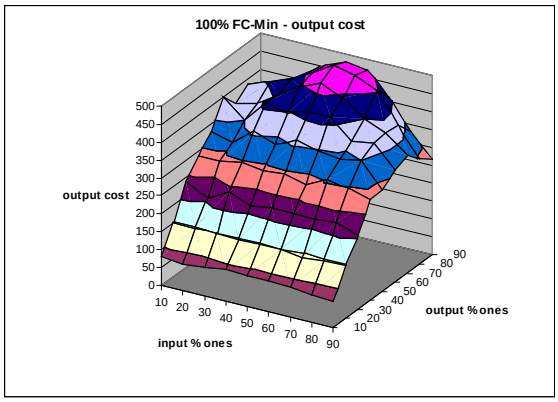
Obrázek 4.5.9 Test č.4 50% BOOM - terms

Tyto grafy vycházejí poměrně shodně. Zajímavostí je, že FC-Min není vůbec citlivý na změnu počtu jedniček na vstupu, což si vysvětluji jeho obráceným postupem při tvorbě implikantů. Ve velké většině je také z tohoto hlediska lepší nebo alespoň stejně dobrý nežli BOOM. BOOM získává náskok pouze při vstupu 10% jedniček respektivě 90% při výstupu mezi 30% až 70% jedniček. Minimalizátor běžící v poměru 50% BOOM a 50% FC-Min vykazuje velmi dobrou aproximaci mezi oběma výsledky.

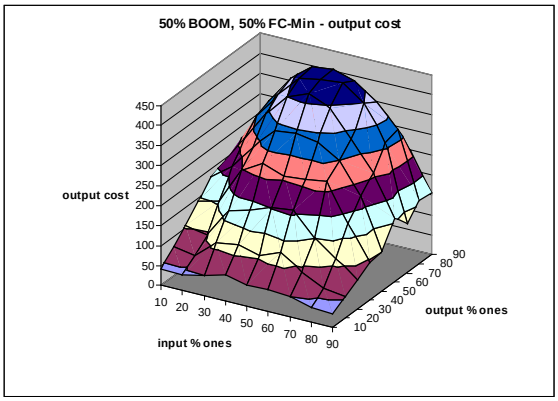
Output cost



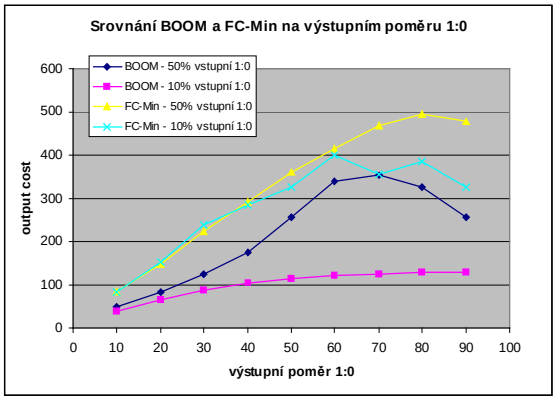
Obrázek 4.5.10 Test č.4 100% BOOM - output cost



Obrázek 4.5.11 Test č.4 100% FC-Min - output cost

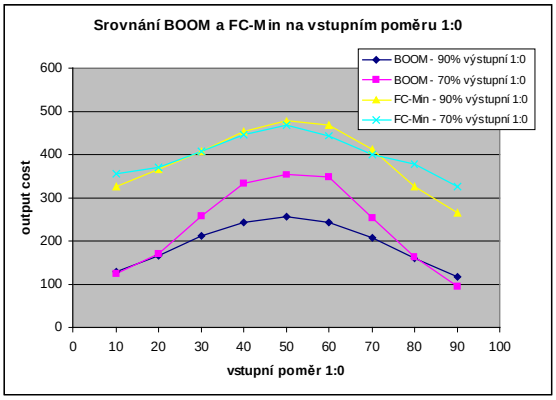


Obrázek 4.5.12 Test č.4 50% BOOM - output cost



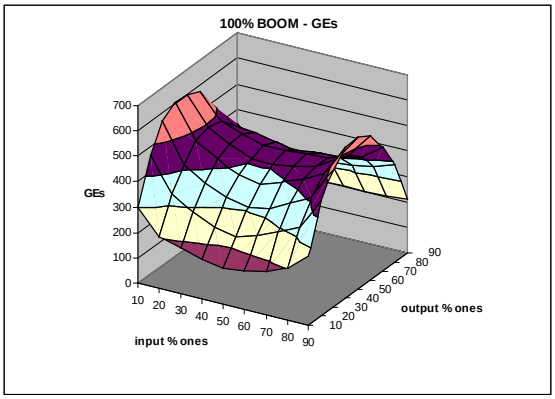
Obrázek 4.5.13 Test č.4 Srovnání - output cost

Tyto grafy považuji za nejvydařenější z celého testu č.4. Oba dva minimalizátory měli maximum, když vstup byl rovnoměrně vyvážen a výstup měl 70% jedniček. To je vzhledem ke kritéiu logické. Zatímco však BOOM se k vrcholu přibližoval pozvolněji spíše konvexně, tak FC-Min přesně naopak s tendencí dosti konkávní. Zase bych si to vysvětlil rozdílným způsobem tvoření skupinových implikantů. V celém rozsahu BOOM překonává FC-Min. Má menší output cost mezi 33% až 80% vztaženo k výsledkům FC-Minu. Kombinace obou minimalizátorů, věrně2 průměruje jejich výsledky.

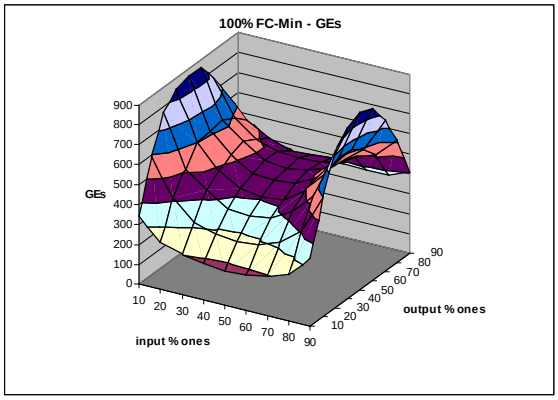


Obrázek 4.5.14 Test č.4 Srovnání - output cost

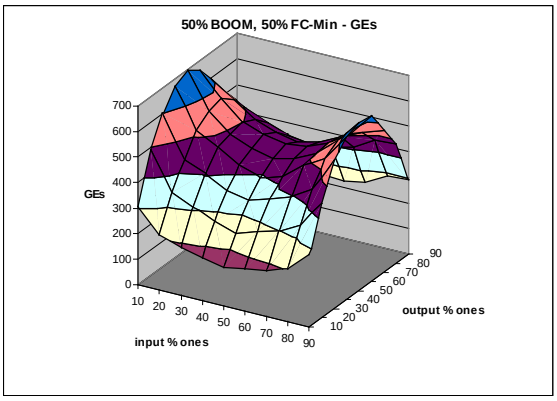
GEs



Obrázek 4.5.15 Test č.4 100% BOOM - GEs



Obrázek 4.5.16 Test č.4 100% FC-Min - GEs



Obrázek 4.5.17 Test č.4 50% BOOM - GEs

I zde podobně jako v kritériu literálů v okrajových podmínkách vstupů byl BOOM výrazně lepší nežli FC-Min. Zajímavé také je že BOOM při velkém počtu jedniček na výstupu (90%) již nebyla závislost výstupu na počtu jedniček na vstupu. Další zajímavostí je, že podobně jako BOOM u literálů, zde FC-Min vykazoval jistou odchylku v případě většího poměru jedniček na výstupu (60+%). FC-Min byl v tomto případě horší nežli v případě malého počtu jedniček na výstupu. I toto nabízí jisté zamyšlení nad využitím této skutečnosti.

Ještě je zajímavé že pouze u tohoto testu se grafy GEs a output cost kategoricky lišily.

4.5.3 Závěr

Skoro ve všech případech byl zde lepší algoritmus BOOM nežli FC-Min, což je ovšem zaplaceno velkou časovou náročností. Bylo to pravděpodobně způsobeno nastavením počtu vstupů a výstupů. Myslím že se v tomto ukázala velmi zajímavá věc a to rozdílné chování minimalizátorů na různé poměry jedniček především na výstupu. Tato problematika by si zajisté žádala další důkladnější testování. A případné využití zjištěných závislostí na zvýšení efektivity těchto minimalizátorů. Takže pokud hledáme minimalní počet hradel a nezáleží na čase, volil bych jednoznačně BOOM, v případě rychlosti je mnohem výkonější minimalizátor FC-Min. Rozumný kompromis mezi oběma variantama nabízí nastavení kombinace obou dvou.

4.6 Test č.5 závislost vstupní granularity don't cares průměrným poměrem 50% a závislost výstupní granularity 1:0 s průměrným poměrem 50%

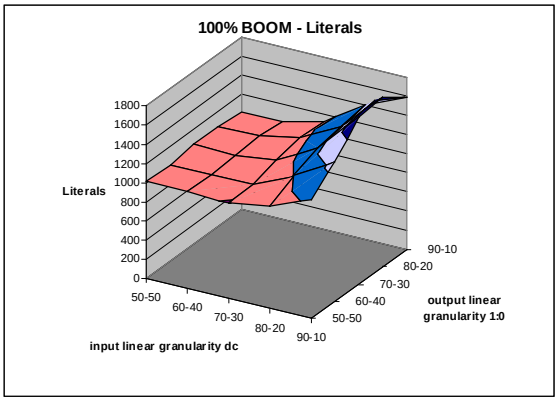
4.6.1 Popis

V tomto testu jsem chtěl otestovat další už poměrně složitěji hledatelné rozdíly ve vstupních tabulkách.

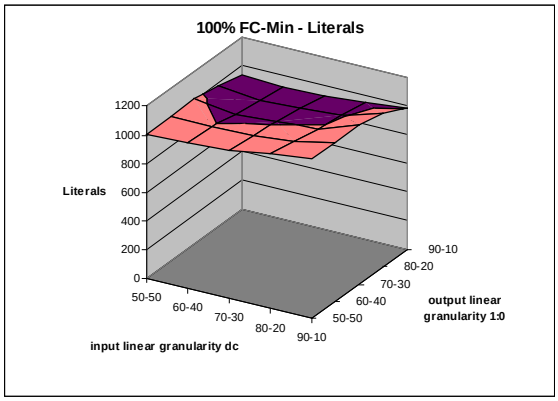
Rozsah vstupní granularity don't care byl nastaven v pěti krocích a v každém byla průměrná granularita 50%. Kroky teda jsou: 90-10, 80-20, 70-30, 60-40 a 50-50. například granularita 1:0 80-35 znemená že na prvním termu bude 80% jedniček lineárně klesající až do posledního termu kde bude jen 35% jeniček. Taktéž byla zvolena granularita 1:0 na výstupu

4.6.2 Výsledky

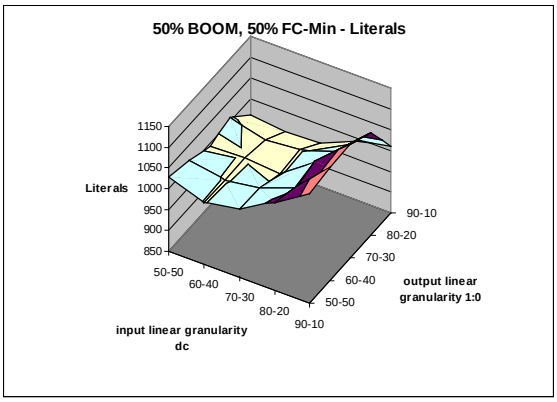
Literals



Obrázek 4.6.1 Test č.5 100% BOOM - Literals



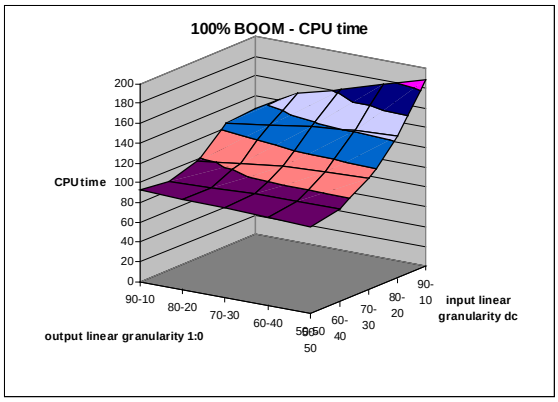
Obrázek 4.6.2 Test č.5 100% FC-Min - Literals



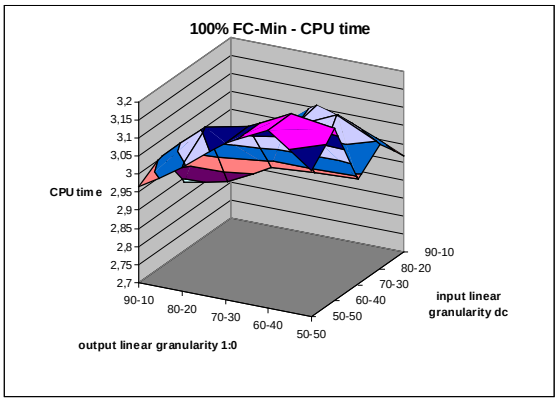
Obrázek 4.6.3 Test č.5 50% BOOM - Literals

Je videt že v případě literálů nemá ani jedna granularita vliv na minimalizátor FC-Min. BOOM je ovlivněn vstupní granulairtou don't care, při zvyšující se nerovnoměrném rozložení jedniček na vstupu. Zhoršuje výsledky až o 80%.

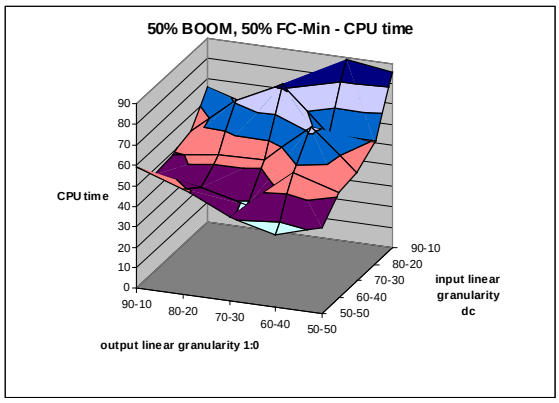
CPU time



Obrázek 4.6.4 Test č.5 100% BOOM - CPU time



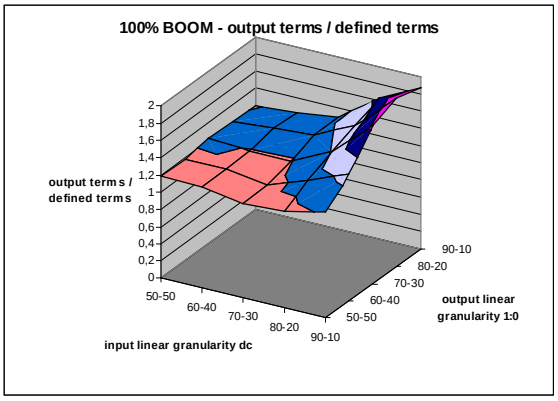
Obrázek 4.6.5 Test č.5 100% FC-Min - CPU time



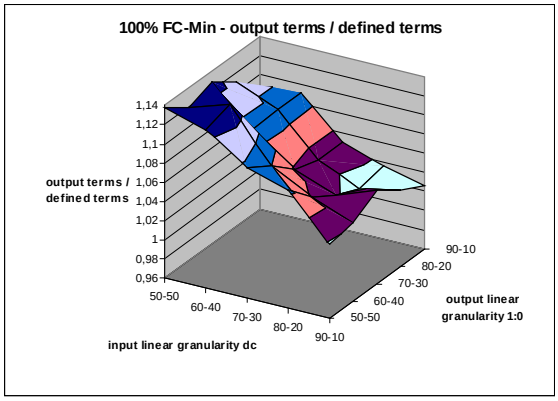
Obrázek 4.6.6 Test č.5 50% BOOM - CPU time

V čase opět je výrazně lepší FC-Min a to až o dva řády. FC-Min byl velmi nevyrovnaný ve výsledcích a vykazoval náhodné odchylky spíše než závislosti na proměnných. Bylo to pravděpodobně velmi krátkým časem který minimalizátor běžel. Zatímco BOOM vykazoval závislost a to, že při velké granularitě don't care na vstupech pracuje až 2krát pomaleji. Částečně to zlepšuje vysoká granularita 1:0 na výstupu.

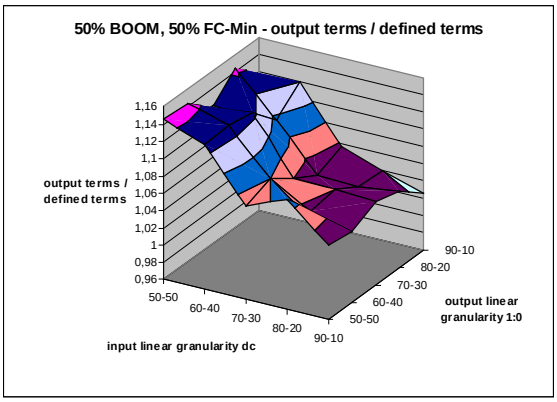
Output terms / defined terms



Obrázek 4.6.7 Test č.5 100% BOOM - terms



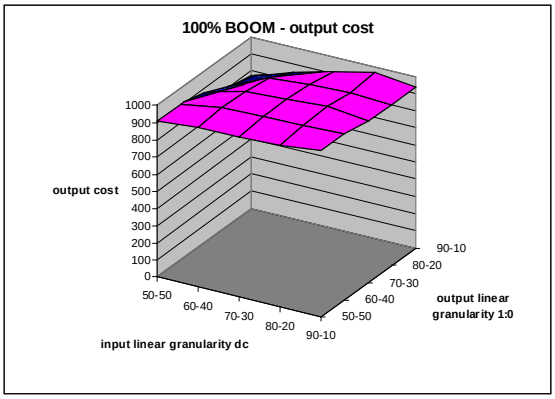
Obrázek 4.6.8 Test č.5 100% FC-Min - terms



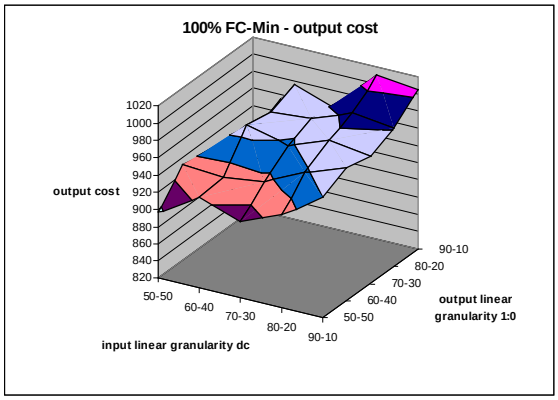
Obrázek 4.6.9 Test č.5 50% BOOM - terms

Zde opět BOOM vykazuje jistou závislost na granularitě vstupních don't care a to že stoupá počet výstupních termů se zvyšující se granularitou a to až skoro na dvojnásobek. FC-Min naopak zde vykazuje opačnou tendenci a to, že se rostoucí granularitou don't care na vstupu klesá počet termů, ale pouze o nějakých 15%. Závislost na granularitě 1:0 na výstupu nebyla nalezena.

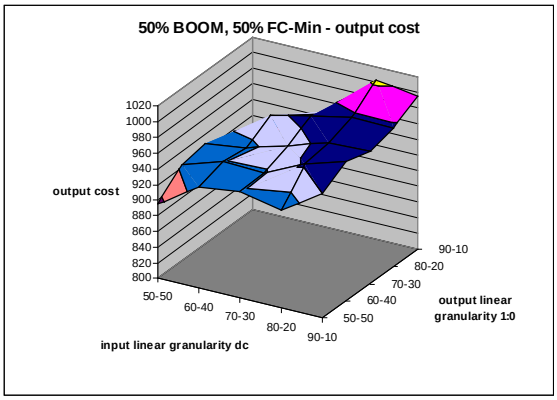
Output cost



Obrázek 4.6.10 Test č.5 100% BOOM - output cost



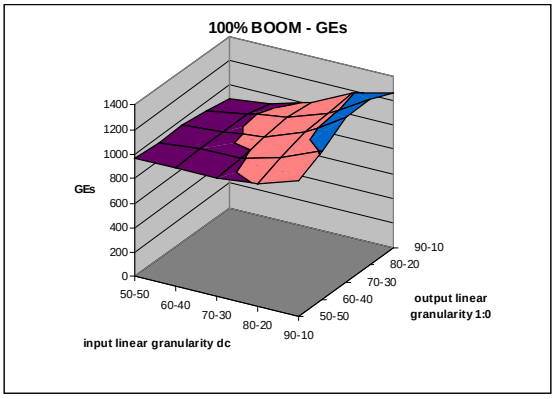
Obrázek 4.6.11 Test č.5 100% FC-Min - output cost



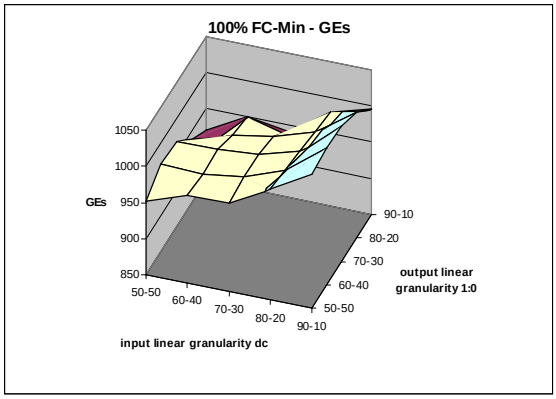
Obrázek 4.6.12 Test č.5 50% BOOM - output cost

Minimalizátor BOOM se v tomto kritériu chová nezávisle na žádné měněné proměnné. FC-Min podobně jako v kríteriu čas vrací dosti nevyrovnané výsledky s velkou rolí náhody. Avšak jistá malá závislost v output cost tu je. Nejhorší výsledky vrací při velké granularitě na vstupních don't cares zároveň s velkou granularitou 1:0 na výstupu, tato odchylka činí přibližně 10%.

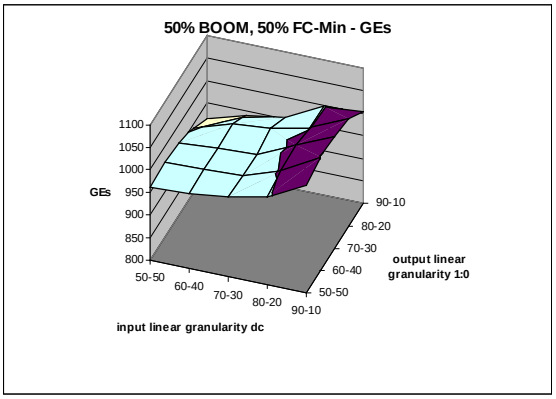
GEs



Obrázek 4.6.13 Test č.5 100% BOOM - GEs



Obrázek 4.6.14 Test č.5 100% FC-Min - GEs



Obrázek 4.6.15 Test č.5 50% BOOM - GEs

U BOOMu se s rostoucí granularitou vstupních don't cares zhoršuje výsledek a to až o 20%. Podobně FC-min také ztrácel s velkou granularitou, ale pouze přibližně 10%. Jinak oba dva minimalizátory vykazovaly podobné výsledky a nelze určit který byl lepší.

4.6.3 Závěr

Tento test ukázal, že granularita nikterak výrazně neovlivňuje chování obou minimalizátorů. Pouze BOOM pracuje pomaleji a s menší jakostí veškerých výstupů při granularitách don't care na vstupu větších než 80-20. Oba dva minimalizátory podávali srovnatelně kvalitní výstup ve všech kategoriích kromě času a počtu termů. V čase byl FC-Min skoro o 2 řády rychlejší a generoval o 20% až 100% méně termů. Opět nastavení poměru spolupráce BOOM a FC-Min vedlo k očekávanému správnému kompromisu mezi oběma variantama.

4.7 Celkový závěr z testů

Myslím že všechny testy dopadly vcelku podle očekávání. Ve většině testovaných případů dopadl lépe minimalizátor BOOM, ovšem s řádově delšími výpočetními časy. Naopak FC-Min je neuvěřitelně rychlý minimalizátor, který se přibližuje výsledku už od první iterace. Ve velké většině testů kombinace obou dvou minimalizátorů věrně průměrovala výsledky každého zvlášť. Domnívám se že při větším počtu iterací by se měla spíše blížit lepšímu výsledku než pouze průměrovat BOOM a FC-Min.

Nejzajímavější bylo testování poměrně specifických vlastností jako je vliv grnularity na výsledek. Občas to přineslo zajímavé náznaky, které by stály za další zkoumání a testování. Jmenovitě například v testu číslo 4. kde se zkoumal vliv různého poměru jedniček a nul jak na vstupu tak na výstupu. Zde docházelo, že BOOM při vetším počtu jedniček než nul na výstupu pracoval lépe než obraceném poměru.

Zklamáním naopak bylo pro mne to, že granularita nepřinesla skoro žádný vliv na testované minimalizátory.

5 Seznam literatury

[1] W.V. Quine: The problem of simplifying truth functions, Amer. Math. Monthly, 59, No.8, 1952, pp. 521-531

[2] E.J. McCluskey: Minimization of Boolean functions, The Bell System Technical Journal, 35, No.5, Nov. 1956, pp. 1417-1444

[3] J. Hlavička and P. Fišer, "BOOM - a Heuristic Boolean Minimizer", Proc. ICCAD-2001, San Jose, Cal. (USA), 4.-8.11.2001, 439-442

[4] P. Fišer, J. Hlavička a H. Kubátová, "FC-Min: A Fast Multi-Output Boolean Minimizer", Proc. Euromicro Symposium on Digital Systems Design (DSD'03), Antalya (TR), 3.-5.9.2003, pp. 451-451

[5] P. Fišer, H. Kubátová, "Two-Level Boolean Minimizer BOOM-II", Proc. 6th Int. Workshop on Boolean Problems (IWSBP'04), Freiberg, Germany, 23.-24.9.2004, pp. 221-228

[6] Coudert, O.: Two-Level Logic Minimization: An Overview, Integration. The VLSI Journal, 17-2, pp. 97-140, Oct. 1994.

[7] V. Jáneš, J. Douša, "Logické systémy", skriptum ČVUT Praha, 2001

[8] http://cs.felk.cvut.cz/~fiserp/projects.html

A. Uživatelská příručka

Syntaxe:

generate -i n -o n -t n [-h n] [-idc n] [-odc n] [-odt n] [-idt n] [-glx n] [-IN] [-NC] [-L] [-timeout n] [file_name]

-i *n*	Nastav počet vstupních proměnných na *n.*
-o *n*	Nastav počet výstupních proměnných na *n.*
-t *n*	Nastav počet generovaných termů na *n*.
-h *n*	Nastav počet bitů použítých v hashovací funkci na *n*. Standartní hodnota je 0. Maximální hodnota je 16.
-idc *n*	Nastav počet don't cares ve vstupní matici na *n* %. Standartní hodnota je 0%.
-odc *n*	Nastav počet don't cares ve výstupní matici na *n* %. Standartní hodnota je 0%.
-idt *n*	Nastav poměr 1:0 ve vstupní matici na *n* %. Standartní hodnota je 50%.
-odc *n*	Nastav poměr 1:0 ve výstupní matici na *n* %. Standartní hodnota je 50%.
-glx *n*	Nastav lineární granularitu, za x lze doplnit idc - nastav druhou mez počtu don't cares ve vstupní matici na *n* %. odc - nastav druhou mez počtu don't cares ve výstupní matici na *n* %. idt - nastav druhou mez poměru 1:0 ve vstupní matici na *n* %. odt - nastav druhou mez poměru 1:0 ve výstupní matici na *n* %. při použití bude první generovaný term mít valstnost uvedenou v x, poslední term bude mít vlastnost uvedenou v glx.
-IN	Vypiš informace o funkci.
-NC	Neprovádí se kontrola konzistence a generuje se PLA typu fd.
-L	Nahraj tabulku uvedenou ve file_name a vypiš o ní info.
-timeout *n*	Nastav timeout na *n* sekund. Standartní hodnota je 1000 s
file_name	výstupní soubor. Pokud není specifikován PLA tabulka je vypsána na standartní výstup.

Tabulka A.1 Popis přepínačů programu generate

Program generuje náhodnou booleovskou funkci ve formátu PLA typu fr nebo typu fd. Standartně je generována tabulka typu fd a tudíž je prováděn test konzistence. Aby nedošlo zařazení některého z termů zaroveň do on-setu a off-setu. Tímto při generování funkcí s velkým procentem vstupních don't care velmi silně stoupá spotřebovaný výpočetní výkon. Proto je v programu nastaven timeout standartně na 1000 vteřin po kterých se program ukončí bez výsledného výstupu.

Příklady

generate -i 100 -o 20 -t 50 -idc 20 out.pla

Generuje PLA tabulku typu fd se 100 vstupními proměnnými, s 20ti výstupními proměnnými. Tabulka bude mít 50 termů a 20% don't cares na vstupu. Tabulka se uloží do souboru out.pla

generate -i 50 -o 20 -t 100 -idt 75 -glidt 25 -IN

Generuje tabulku na standartní výstup s 50ti vstupními proměnnými, se 100 termy a s 20ti výstupními proměnnými. První generovaný term bude mít 75% jedniček na vstupu. Poslední generovaný jich bude mít 25%, zbytek budou nuly. Po ukončeníse na standartní výstup vytisknou statistické informace o vygenerované PLA tabulce.

Poznámka

Při generování lineární granularity don't cares je mnohem výhodnější ji zadávat na vstupní matici sestupně. Např je lepší volit přepínače -idc 80 -glidc 20, nežli opačně. Na výstupní matici je tomu obráceně - je lepší volit menší procento u přepínače -odc.

B. Obsah CD

V kořenovém adresáři se nachází binární spustitelný soubor generate.exe a podadresáře:

src - zde jsou uloženy zdrojové soubory ke generátoru

texty - složka obsahuje tuto práci v elektronické podobě

testy - obsahuje veškeré materiály týkající se provedených testů včetně všech vstupních dat, skriptů na převod formátů, výstupních dat a grafů

České vysoké učení technické v Praze

Fakulta elektrotechnická

Bakalářská práce

Parametrizovaný generátor náhodných booleovských funkcí

Tomáš Měchura

Vedoucí práce: Ing. Petr Fišer

Studijní program: Informatika a výpočetní technika

leden 2006

Poděkování

Rád bych poděkoval panu Ing. Petru Fišerovi za ochotu a vůli při pomoci s vypracováním a dokončením této práce.

Prohlášení

Prohlašuji, že jsem svou bakalářskou 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 orávem autorským a o změně některých zákonů (autorský zákon)

V Praze dne 27.1.2005 .....

Abstract

This work describes the making of generator of boolean function in th form of PLA table. It tries to find effective solution of generating large number overlaping terms. The second part delas with suitable testing of boolean minimizer BOOM. The tests are done with many views on various parameters of minimalized function. From these tests it tries to make some conlcusions, which method of minimalization is better for partivular parameter.

Anotace

Tato práce pojednává o tvorbě generátoru náhodných booleovských funkcí formou PLA tabulky. Snaží se najít vhodná efektivní řešení při generování velkého počtu překrývajících se termů. Druhá část práce se zabývá vhodným testováním minimalizátoru BOOM-II. Testování probíhá z mnoha různých pohledů na parametry minimalizované funkce. Z těchto testů se pak snaží vyvodit závěry, která minimalizační metoda je lepší pro dané parametry booleovské funkce.

Obsah

Seznam obrázků

Seznam tabulek

Slovník pojmů

Literál - proměnná nebo její negace

Term - Term je vyjádřením součtu nebo součinu literálů. Pokud se jedná o součin, jedná se o součinový term, neboli P-term. Jedná-li se o součet, nazveme jej součtový term nebo-li S-term.

Minterm - součinový term, který obsahuje všechny vstupní proměnné. pro n vstupních proměnných existuje 2^n různých mintermů.

Maxterm - součtový term, který obsahuje všechny vstupní proměnné.

on-set - stav proměnných logické funkce, ve kterých je funkční hodnota rovna 1.

off-set - stav proměnných logické funkce, ve kterých je funkční hodnota rovna 0.

dc-set - stav proměnných logické funkce, na kterých daná fuknce nezávisí.

don't care - označení proměnné v PLA tabulce, které znamená, že daný term na této proměnné nezávisí.

Implikant logické funkce - Jedná se o výraz ve tvaru P-termu, pro který platí, že danou funkci implikuje; tzn. jestliže nabývá hodnoty logické 1, daná funkce nabývá též hodnoty logické 1. Implikant nazveme přímým implikantem právě tehdy, když po vypuštění libovolného literálu přestává být implikantem.

Iterativní minimalizace - minalizace založená na skutečnosti, že některá část minimalizačního procesu je řízena náhodou. Takže pokud algoritmus spustíme vícekrát na stejný problém nemusí vždy vést ke stejnému výsledku. Navíc, kombinací vygenerovaných implikantů z jednotlivých běhů iteračního procesu minimalizace, lze dosáhnou výsledků lepších

1 Úvod

Cílem této práce bylo vytvořit parametrizovaný generátor náhodných booleovských funkcí s výstupem formou PLA tabulky. A poté pomocí jím generovaných tabulek s různými parametry otestovat citlivost některých booleovských minimalizátorů. Nejprve jsem považoval celou úlohu za velmi jednoduchou, ale postupem času jak jsem se více a více dostával do nitra problému generování booleovských funkcí došel jsem k závěru, že vše není tak jednoduché jak se na první pohled zdá.

Na začátku mi p. Fišer poskytl jeho jednoduchý generátor, na kterém jsem se seznámil s vyskytujícími se problémy při generování. Tyto problémy jsem se snažil v této práci vyřešit, případně omezit různými metodami, což bylo na celé práci nejtěžší a nejvíce frustrující, poněvadž některé techniky byly nakonec silně kontraproduktivní. Pro můj generátor jsem se rozhodl zefektivnit a doplnit o potřebné funkce již zmíněný generátor p.Fišera, radši než začínat tvořit úplně nový. Avšak v průběhu implementování jsem díky novým vlastnostem musel podstatnou část hlavního algoritmu přepsat, takže to nakonec vyšlo nastejno.

Další část mé práce bylo ozkoušet vliv generovaných boolovských funkcí s různými parametry na výsledky vybraných minimalizátorů. Z čehož pak udělat jakýsi závěr, v jakých připadech je lépe použít jaký druh minimalizátoru. Poněvadž jsem s minimalizátory doposud neměl žádné zkušenosti. Tak mě tato část práce velmi lákala i přesto, že jsem si byl vědom hrozné otročí práce na přípravě testů a zpracovávání výsledků. Počátkem testovací fáze práce jsem se dozvěděl že p. Fišer je autor projektu BOOM-II a že by ho tímto způsobem rád otestoval.

1.1 Požadavky na generátor

Zadané požadavky na generátor se mi zdály poměrně snadné i když to byla menší část celé bakalářské práce.

- schopnost generovat PLA tabulky typu fd a typu fr.
- počet vstupů, výstupů a počet termů řádově do tisíce
- možnost nastavení poměru generování don’t cares, jedniček a nul na vstupu a výstupu
- možnost nastavení jisté granularity u generovaných don‘t cares, jedniček a nul jak na vstupu tak výstupu
- schopnost zpětně zjistit tyto poměry z již vygenerované PLA tabulky.
- dokumentace ke generátoru

1.2 Požadavky na testování

V testování byl kladen důraz na ucelenost testů a jejich správné vyhodnocení, především jednoznačně určit, který z minimalizátorů je na určitou PLA tabulku vhodnější.

- otestovat vygenerované PLA tabulky typu fr v dvou různých booleovských minimalizátorech. Zaměřit se především na různé parametry PLA tabulek
- vyzkoušet kombinace nastavení generovaných PLA tabulek, z toho určit zajímavé testy, které po té uskutečnit.
- výsledky testů vyhodnotit z co nejvíce různých hledisek

2 Generátor

Hned v úvodu práce mi p. Fišer nabídl pokračovat v jeho rozdělaném generátoru a poskytl mi zdrojové soubory. Zároveň mi upřesnil požadované parametry výsledného generátoru a hlavně mi vylíčil vyskytující se problém s překrývajícími se termy. Byl jsem v zásadě rád, poněvadž jsem měl na čem zakládat. Přesto jsem si uvědomoval, že některé doimplementovávané funkce to mohou udělat mnohem těžší, než kdyby se s nimi počítalo od začátku. V zadání bakalářské práce je zmiňována implementace vícehodnotové logiky. Po diskusi s p. Fišerem jsme se dopracovali k závěru, že to nebude potřeba, poněvadž každá vícehodnotová logika lze rozepsat na více binárních členů. Generátor měl generovat výstup formou PLA tabulky, typu fd a fr.

formát PLA

PLA tabulka se skládá z klíčových slov počínající tečkou a 1 alfanumerické matice kde každý řádek odpovídá jednomu termu a každý sloupeček odpovídá jedné vstupní nebo výstupní proměnné. Ve směru zleva do prava jsou nejdřív uvedeny všechny vstupní proměnné, až po té všechny výstupní. Symboly užité ve vstupní části matice:

- "0" značí že P-term obsahuje negovaný korespondující literál
- "1" značí že P-term obsahuje korespondující literál.
- "-" značí že P-term korespondující literál vůbec neobsahuje. tzv. don't care

Booleovská funkce je kompletně popsána trojicí on-set, off-set a dc-set. PLA tabulka používa čtyři typy různého zápisu funkce, uvedu zde pouze dva, které byly výstupem generátoru:

- typ fd - funkce je zadána množinou on-set a dc-set. Off-set je vypočitán jako doplněk jejich sjednocení.
- typ fr - funkce je zadána on-setem a off-setem. Dc-set je dopočítán jako doplněk jejich sjednocení.

Symboly užité ve výstupní části matice uvádím v tabulce 2.1:

	typ PLA	
	fd	fr
"1"	on-set	on-set
"0"	nemá význam	off-set
"-"	dc-set	nemá význam
"~"	nemá význam	nemá význam

Tabulka 2.1 význam symbolů ve výstupní části matice

Kompletní definice PLA formátu je k nalezení v [13].

2.1 Analýza problémů

Celý problém v generování booleovských funkcí byl pouze u výstupu PLA tabulky typu fr. Při náhodném vygenerování termu, který se s některým dříve vygenerovaným termem alespoň částečně překrýval, nesmělo dojít u kterékoli výstupní proměnné aby jeden term byl on-set a druhý off-set. Tato situace nastávala až při velkém procentu vygenerovaných mintermů obsažených v on-setu nebo off-setu oproti všem možným mintermům. Příklad:

Máme 10 vstupních proměnných, 2 výstupní, chceme generovat 1100 termů a vstupní procento don't cares je 0. To značí že budeme generovat přímo mintermy. Avšak počet všech možných mintermů je 2in kde in je počet vstupních proměnných, což je v tomto případě 1024, znamená to že minimálně dojde k 76 kolizím. Zde roste pravděpodobnost kolize lineárně.

Horší je to pokud máme jisté vstupní don't cares. Příklad:

Máme 10 vstupních proměnných, 2 výstupní, budeme generovat 300 termů a vstupní procento don't care je 20. Znamená to že budeme generovat termy kde nebudou obsaženy průměrně 20% * 10=2 literály. Takže tento term bude stát za čtyři mintermy. V případech se vstupními don't cares je pravděbodosnost kolize velmi obtížně vyjádřitelná. V nejlepším případě dojde zde ke kolizi po $2^{in(1-dc)}$ vygenerovaných termech.

Avšak pokud dojde ke kolizi neznamená to, že vygenerovaný term nepatří do výsledku. Pokud se ve všech výstupech se všemi s ním kolizními termy nevyskytuje jeden term v on-setu a druhý zároveň v off-setu pro stejnou výstupní proměnnou, tak je term vpořádku a je umístěn do výstupu. Při nulovém procentu výstupních don´t cares pravděpodbnost, že dva termy budou mít stejný výstup klesá exponenciálně s rostoucím počtem výstupů. Pokud zvýšíme don't cares na výstupu pravděpodobnost shody stoupá. Takže don't cares na výstupu začnou pomáhát při generování, až s počátkem kolizí generovaných termů.

2.2 Návrhy a řešení

2.2.1 Popis celého algoritmu.

Celý algortimus je velmi jednoduchý. Postupně se náhodně generuje term s danými pravděpodobnostmi don't cares a s daným poměrem jedniček a nul. Tyto pravděpodobnosti se pro každý term vypočítávají, aby se správně generovala nastavená granularita. Pokud generujeme tabulku typu fd tak se následující kontrola konzistence neprovádí a term se zapisuje normálně do výsledku.

V případě výstupu tabulky PLA typu fd se dělá test konzistence, aby ve funkci nebyl nějaký minterm který by byl zahrnut v on-setu a zároveň v off-setu. Po vygenerování náhodného termu, se hledají v hashtabulce možné kolizní termy, které se pak jeden po druhém porovnávají na překrytí s vygenerovaným termem. Pokud dojde k překrytí termu, tak se začnou porovnávat výstupní proměnné. Zde se kontroluje koznistence termu a to tím způsobem, že pokud jeden term má na výstupu v dané proměnné jedničku a druhý nulu, tak se prohlásí za nekonzistentní. Poté je přepsán jiným náhodným termem a znovu kontrolován s ostatními od začátku. Pokud je určen za konzistentní přidá se term do výsledku a zároveň se správně zařadí do hashtabulky. Pak se celý proces opakuje. Po úspěšném vygenerování počtu zadaných termů se výsledná tabulka uloží a pokud byl požadován výpis informace o PLA , spustí se algoritmus, který celý výsledek projde term po termu a na závěr vyhodnotí skutečná procenta don't cares, poměry 1:0, a metodou lieární regrese vypočítá pravděpodobnou granularitu těchto údajů. Pokud se ovšem zadaný počet termů nevytvoří do nastaveného času, tak se program ukončí.

2.2.2 Heuristika generování termů

Hned z počátku se nabízelo několik možností řešit problém kolizních termů, aby se zvedla pravděpodobnost úspěšného vygenerování konzistentního termu. Všechny se zakládaly na tom, že po několika po sobě jdoucích kolizích, se použije jedna z metod

- negenerovat vstupní proměnné náhodně, ale pokusit se vygenerovat s jistou náhodou nekolizní term nebo alespoň term s menším překryvem s ostatními termy
- vygenerovat náhodný term a zkusit najít nastavení výstupních proměnných tak, aby term mohl být zařazen do výsledku.

Po prodiskutovávání s vedoucím projektu p. Fišerem, byla druhá varianta zavrhnuta, poněvadž by vedla k jistému ovlivnění náhodnosti generované funkce. Proto jsem se zaměřil na první variantu. Tato varianta předpokládala, jistou datovou strukturu, ve které si budu

uchovávat nepoužité termy. Protože měl vyvýjený generátor generovat funkce s řádově tísícem vstupních proměnných, takováto datová struktura nebyla proveditelná, kvůli exponenciální závislosti počtu termů na počtu vstupních proměnných.

Další můj návrh byl použít jako datovou strukturu hash tabulku, kde hashovací funkce byla nastavitelný počet vstupních proměnných. V hash tabulce by se v jednotlivých položkách uchovával spojový seznam odpovídajících již vygenerovaných termů a počet termů v daném seznamu. V případě že by ve vstupní proměnné byl don't care tak by se předpokládaly obě možnosti a přidal by se term do hash tabulky vícekrát. Po rozboru této metody a zjištění že by při větších vstupních don't cares byla kontraproduktivní jsem metodu zavrhl. Heuristiku jsme totiž. potřebovali především pro funkce s velkým vstupním procentem don't cares. Poněvadž jsem další nápady na heuristiku neměl, soustředil jsem se na zefektivnění ostatních částí programu.

2.2.3 Zefektivnění hledání a vyhodnocování inkozistencí

Strukturu hash tabulky jsem nezavrhl a implementoval jsem jí do části programu, který hledá kolize. K danému termu najde v hash tabulce všechny adepty na kolizi a hledá jí tak v omezeném množství termů. Toto ovšem neplatí, stejně jako v předchozím případě, pro větší množství vstupních don't cares. Z předpokladu abychom neprocházeli větší množství termů s hash tabulkou než bez hash tabulky nám vychází nerovnost ze které určíme pro jaké procento don't cares by měla být tabulka teoreticky přínosná:

b - počet vstupních proměnných, které bere v úvahu hashovací funkce.

dc - číslo udávající poměr don't cares na vstupu

t - počet vygenerovaných termů´

Počet termů v celé hash tabulce ht je roven počet termů t krát průměrná dimenze termu tvořeného pouze z prvních b vstupních proměnných.

$$ht = 2^{dc*b} * t$$

Počet termů v jedné položce tabulky pt je počet všech termů v hashtabulce lomeno počtem ploţek

$$pt = \frac{ht}{2^b} = 2^{(dc-1)*b} * t$$

Počet termů ve kterých se bude hledat při použití tabulky je roven součinu pt a průměrnému počtu položek, ve kterých se bude kolize hledat. A ten musí být menší než počet termů. Z toho nám vyjde očekávaná hranice dc.

$$pt * 2^{dc*b} = 2^{(2dc-1)*b} * t < t$$

$$2^{(2dc-1)*b} < 1$$

$$(2dc-1) * b < 0$$

$$dc < 0{,}5$$

Výsledná teoretická hranice 50% vstupních don't cares nebyla v testech potvrzena, pravděpodobně díky větším nárokům při hledání v tabulce.

Hash tabulka byla implementována pomocí pole ukazatelů na jednotlivé položky. Každá položka byla spojový seznam ukazatelů na termy. Tím docházelo k šetření místa, poněvadž jsme tentýž term neukládaly vícekrát.

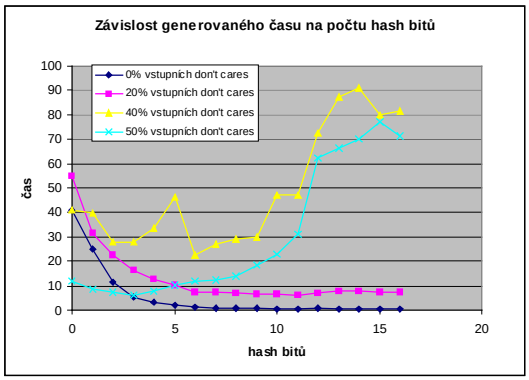
2.2.4 Ostatní vylepšení

Další zlepšení, které jsem implementoval, spočívá ve zkoušení vygenerování více kombinací výstupních proměnných pro jednu kolizní funkci. Toto přineslo nepatrné výsledky pouze v případech kde pomáhá i hash tabulka. Takže je v generátoru standartně nastavena na jedno opakování a není tato vlastnost dokumentovaná v příručce.

Po domluvě jsem do programu implementoval lineární granularitu. Původně jsem uvažoval i o dalších typech jako například kvadratická apod. Dále jsem ještě uvažoval o nějakém algoritmu který by průběžně kontroloval a zaručoval aby výstup měl opravdu požadované kvality především u vstupních don't cares. Návrh byl takový že algortimus bude průběžně vypočítávat zatím vygenerované statistiky a podle toho bude mírně pozměňovat vlastnosti dalšího generovaného termu, aby korigoval vznikající odchylky. Generátor však vykazoval chyby v generovaném input dc v průměru kolem 6ti procent, což jsem považoval za adekvátní. V těžších připadech s více output a vstupních don't cares se to zvyšovalo. Avšak v těchto případech by při použití zmiňovaného algoritmu na korekci chyby mohl celý generátor uváznout, protože by vyhovující již term nemusel existovat.

2.3 Rámcový test

Zde jsem testoval čas generovaní booleovské funkce v závislosti na počtu bitů hashovací funkce. Testy proběhly pro různá nastavení vstupních don't cares a to 0%,20%, 40%, 50% a 60%. Chci upozornit že pro každé nastavení don't cares byl jinak nastaven počet termů. Aby se daly křivky srovnávat v jednom grafu. Šlo mi především o porovnání průběhů křivky pro jiná don't cares a tím ověřit skutečnost, že při vyšších don't cares se bude účinnost generování zhoršovat. Podmínky tohoto testu jsou stejné jako byly v testech v odstavci 3.



Obrázek 2.3.1 Závislost času generování na hash bitech

Zde se ukázalo že teoreticky vypočítaná mez vstupního procenta don't cares byla poměrně přesná. Minimum času pro 0% don't cares bylo při použití 14ti hashbitů. Při 20% don't cares bylo minimum kolem 10ti hash bitů. Při 50% docházelo zlepšení pouze do 4 bitů.

Z toho vyplývá že lze úspěšně hash tabulku použít, ale musí se vědět kdy a v jakých případech. Při generování do 20% vstupních don't cares se může nastavit 10 bitů, při generování do 50% bych nechal nastavení do 5ti bitů. Pro větší procenta je již hash tabulka kontraproduktivní.

2.4 Závěr

Myslím si že se mi podařilo vytvořit užitečný nástroj na generování booleovských funkcí ve formě PLA tabulky, který je dostatečně výkonný a nabízí dost možností nastavení parametrů generované funkce. Program lze úspěšně využít pro generování booleovských funkcí do souboru testů vhodných pro testování booleovských minimalizátorů. Toto jsem úspěšně ověřil v druhé stěžejní části této práce.

3 Minimalizátory

Poté co jsem dokončil práci na generátoru, jsem se mohl začít plně věnovat další části bakalářské práce, která byla testování chování minimalizátorů. Jak již jsem několikrát zmínil, první co se muselo udělat, bylo vybrat alespoň dva různé booleovské minimalizátory, na kterých by proběhlo testování.

3.1 Analýza a výběr minimalizátorů

Na začátku práce mi p. Fišer vyvětlil, že si představuje abych jako minimalizátory použil jeho velmi dobrý BOOM-II. S tím jsem samozřejmě souhlasil. Přesto jsem se ovšem poohlídnul po jiných možných variantách, abych si zjistil co nejvíce o možných způsobech minimalizace a pochopil výhody a nevýhody zvolených minimalizátorů. V tom byl základ pro určení zajímavých testů a správného vyhodnocení výsledků.

Ze zjištěných informací jsem se utvrdil že BOOM-II je správná volba, poněvadž obsahuje 2 různé minimalizátory, které se navzájem doplňují. Kombinací obou dvou je vhodný pro velký počet vstůpních i výstupních proměnných. Ostatní minimalizátory se mi zdály mnohem méně univerzálnější nebo úplně nepoužitelné poněvadž například neumožňovaly vstup v PLA tabulce. Jmenujme jen některé z nich - ESPRESSO, Karnaugh Minimizer, Scherzo.

3.2 Popis funkce minimalizátorů

3.2.1 BOOM-II

BOOM-II je dvouúrovňový booleovský minimalizátor kombinující dva minimalizátory BOOM a FC-Min. Funkčně vychází z minimalizační metody Quine-McCluskey. Minimalizace probíhá obecně ve dvou fázích. První fáze je generování přímých implikantů vstupní funkce. Tato fáze probíhá iterativně a BOOM-II v ní volá podle nastaveného poměru jeden z minimalizátorů. Dochází tím k vyzužití výhod obou dvou minimalizátorů především při tvorbě skupinových přímých implikantů. Po každé iteraci se výsledky sloučí dohromady s předchozími. A dokud není splněno ukončující kritérium, tak program pouští další iterace. Jinak se přechází do další fáze.

Druhá fáze "problém pokrytí" je společná pro oba minimalizátory. Zde dochází k volbě minimální množiny přímých implikantů ze všech výsledků první fáze s ohledem na optimalizační kritérium. BOOM-II nabízí několik přístupů k řešení toho problému. Standartně je používaný algoritmus "Contributive Addition", který pro každý term počítá skóre a nejlépe ohodnocený term přidá do výsledku. To se opakuje dokud nepokryje celou funkci.

BOOM-II bere jako vstup PLA tabulku typu fr. Při spuštění minimalizace lze nastavit poměr v jakém mají oba minimalizátory běžet. Dále kritérium při kterém se program úspěšně ukončí (například po provedení N iterací, nebo uplynutí určitého času). Další důležité nastavení jsou optimalizační kritérium (literály, output cost, termy, atd.), jaký algoritmus bude řešit fázi "problém pokrytí" a v neposlední řadě výstupní formát. Poté se ještě dají volit různé nastavení heuristik a algoritmů specifické pro jednotlivé minimalizátory.

3.2.2 BOOM

BOOM = BOOlean Minimalizer

BOOM je minimalizátor vycházející z klasické Quine-McCluskeyho metody. Tato metoda minimalizace se uskutečňuje ve třech etapách.

V první etapě se nejprve vytvoří hyperkrychle dimenze počtu vstupů a poté se postupně zmenšuje její dimenze přidáváním literálů. Při tomto procesu se využívá heuristiky, která určuje který literál přidat, aby zredukovaná hyperkrychle pokrývala co možná nejvíce 1-termů.. Pokud již hyperkrychle nepokrývá žádný 0-term, pak je tato hyperkrychle implikantem funkce. Pokryté termy vyřadíme a začneme s novou hyperkrychlí. Takto

pokračujeme dokud nevyřadíme všechny vstupní on-set termy. Takto získané impikanty, nemusejí být přímé. Minimalizátor poté přechází do další etapy.

V této etapě "Implicant expansion" se minimalizátor pokouší z jich hotových implikantů odebrat literály, aby dosáhl přímých implikantů. V této fázi se dá využít více různých heuristik pro volbu literálů. Standartně se používá "Multiple sequential implicant expansion", při kterém se zkouší postupně odebírát všechny literály, a každý vytvořený přímý implikant se uloží. Poté se přejde na další implikant z předchozí etapy. Toto se opakuje do vyčerpání všech možností.

Pokud měla funkce více výstupů, opakujeme předchozí dvě etapy pro všechny výstupy zvlášť. Pak přejdeme do fáze "Implikant reduction", kde se ze všech přímých implikantů snaží vytvořit skupinové implikanty přidáváním literálů. Literály se přidávají sekvenčně až do doby, kdy je zřejmé, že daný implikant se nepoužije pro více výstupních funkcí. Tímto získáme pouze přímé implikanty.

3.2.3 FC-Min

Narozdíl od BOOMu a všech klasických minimalizátorů, FC-Min nevychází ze zadaných termů, ale začíná u výstupů, kde hledá shluky. Takže vlastně postupuje odzadu. Průběh algoritmu se dá rozdělit na tři etapy.

První etapa "Find Cover" vystihuje svůj název a dochází v ní k hledání shluků jedniček ve výstupu. Poněvadž je tento problém velmi složitý, dochází zde k jisté heuristice. Nejprve se vybere term který má na výstupech nejvíce nepokrytých jedniček a pokryje se co nejvíce z nich obdelníkem. Poté se přidá další term a snažíme se zvětšit obdelník aby pokrýval více jedniček. Počet takto přidávaných termů je voleno pravděpodobností při spuštění minimalizátoru. Pokud se již nepřidá term, označíme veškeré jedničky v obdelníku za pokryté a celý cyklus výběru termů opakujeme. Tato etapa se ukončí po pokrytí všech jedniček na výstupu.

Další etapa "Find Implicants" hledá k odpovídajícímu pokrytí implikanty. K danému pokrytí se tvoří implikant tak, že vezmeme všechny odpovídající termy k pokrytí a z nich utvořená minimální hyperkrychle je skupinový implikant. Tento způsob aplikujeme na všechna pokrytí postupně.

Další fáze "Input expansion" už se jen snaží rozšířit skupinové implikanty odebíráním literálů do oblastí s don't cares. Tímto dochází k lepším výsledkům minimalizace.aby minimalizovala output cost a jiné výstupní parametry. Pokud již nelze žádný term rozšířit, tak FC-Min končí

3.3 Závěr

Z výše popsaného fungování obou minimalizátorů a z testů uvedených v [13] jasně vyplývá, že by minimalizátor BOOM měl být účinnější na vstupy s mnoha vstupními proměnnými a málo výstupními. Naopak FC-min by měl být účinnější na vstupy s mnoha výstupy a menším počtem vstupů.

4 Testy

4.1 Úvod

4.1.1 Prostředí

V testování jsem použil PLA generátor verze 1.1 z ledna.2006 a BOOM-II verze 2.6 z listopadu 2005. Veškeré testy probíhaly různě na dvou strojích které byly velmi podobné, jak v hardwarovém vybavení tak v softwarovém vybavení. První počítač je o malinko pomalejší v rychlosti, avšak kompenzuje to o něco rychlejšími přístupovými časy do pamětí. V několika předběžných zde neuveřejněných testech jsem si ověřil, zda oba počítače dávájí stejné výsledky v několika různých odlišných nastaveních, aby jejich použití bylo relevantní a výsledky se daly navzájem porovnávat. Musím ještě podotknout, že test jako celek se vždy dělal pouze na jednom stroji. Zde je hrubá konfigurace obou z nich

PC1

Jedná se o procesor AMD Athlon XP+ bežící na frekvenci 1475 MHz, 512 MB RAM DDR 333 MHz, platforma Win2000 SP4.

PC2

Procesor AMD Athlon MP běžící na frekvenci 1575 MHz, 512 MB RAM DDR 333 MHz, platforma Win 2000 SP4.

4.1.2 Obecné nastavení

Veškeré testy se skládaly z více částí a byly vyhotovovány v pěti kopiích. Veškeré výsledky ze všech testů pak byly z těchto odpovídajících kopií z průměrovány a zde jsou udávany pouze tyto upravené výsledky. Toto opatření bylo děláno z důvodu ošetření možnosti statistické chyby.

První část byla generování vstupních PLA tabulek pomocí vytvořeného generátoru. Nastavení přepínačů generátoru bylo necháno na standartní nastavení a pokud se v testu nebyl proměnný vstup, výstup a počet termů tak byl nastaven na 100 vstupů, 20 výstupů, 100 termů, žádné don't cares, poměr jedniček a nul na 50:50, hash tabulka na 0 bitů. Tuto kombinaci vstupů, výstupů a termů jsem volil z předpokladu jistého obecnějšího použití. Předpokládal

jsem že většina funkcí, které jsou potřeba v praxi minimalizovat, má více vstupů než výstupů. Příkaz obecně vypadal takto.

generate_pr -i 100 -o 20 -t 100 -idc 0 -glidc 0 -odc 0 -glodc -idt 50 -glidt 50 -odt 50 -glodt 50 -h 0 in.XX.N.pla > in.XX.N.info

Kde XX bylo proměnlivé nastavení zkoumaných přepínačů a N bylo číselné označení kopie PLA tabulky.

Samotné testování minimalizátorů probíhalo ve všech testech ve třech nastaveních poměru běhu obou z nich a to 100% BOOM, 100% FC-Min a 50% BOOM s 50% FC-Min. Všechny testy pokud není jinak uvedeno probíhaly se základním nastavením všech přepínačů a to především nastavení ukončujícího kritéria na počet iterací 10 a optimalizačního kritéria na output cost+literals.

Nastavení ukončujícího kritéria standartně na 10 iterací bylo způsoben chybou v minimalizátoru BOOM-II. V průběhu minimalizace dochází k tzv. memory leak, který přibližně po 30ti až 50ti minutách začal vést k stále většímu swapování až nastala situace kdy program naprostou většinu času swapoval a již nic efektivně neprováděl. Tato situace se projevila skoro ve všech testech, kdy jsem nejprve musel metodou pokusu a omylu určit přibližné hranice problému, aby nedocházelo k onomu problému se swapováním. Musím podotknout že tento jev se projevoval u časově náročnějších testech, které byly pouze s minimalizátorem BOOM. FC-Min vykazoval řádově rychlejší zpracování výsledku. Příkaz vypadal přibližně takto

boom_pr -Si10 -FR 0 in.XX.N.pla out.0.XX.N.pla 2> out.0.XX.N.info

Kde XX bylo proměnlivé nastavení zkoumaných přepínačů a N bylo číselné označení kopie PLA tabulky. Význam přepínačů lze zjistit v manuálu k aplikaci [].

Vyhodnocování testů probíhalo z co nejvíce možných hledisek, co nám nabízel minimalizátor na výstupu. A to:

- CPU time - doba trvání výpočetního výkonu na dosažení výsledku
- output terms / defined terms - poměr termů po minimalizaci a termů před minimalizací.
- literals - počet lietrálů ve výstupní disjunktivní formě
- output cost - počet vstupů do všech výstupních OR hradel
- GEs - počet hradel potřebné k výrobě obvodu.

4.2 Test č.1 závislost počtu vstupů a počtu termů

4.2.1 Popis

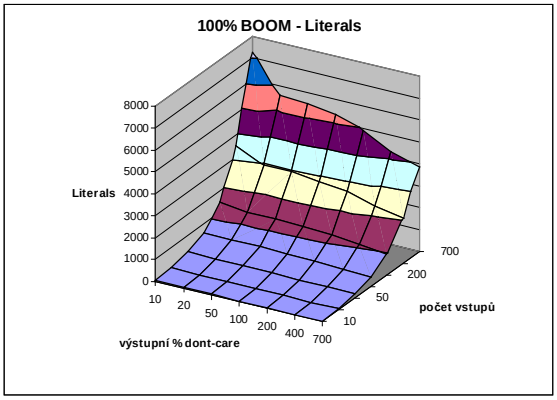
Tímto testem jsem zamýšlel otestovat jak se chovají oba minimalizátory při velkém počtu vstupů s nastaveným počtem výstupů na 20. Spíše to bylo postaveno na otestování FC-Minu při poměru počet vstupů/počet výstupů. Zároveň jsem byl zvědavý jak si poradí s velkým rozdílem počet vstupů a počet termů.

Trojrozměrné grafy uvedené v tomto testu jsou zkreslené. Poněvadž vynášené hodnoty na osách nereprezentují skutečnou vzdálenost od počátku osy. Je to způsobené omezením softwaru a je si třeba na to dát pozor, hlavně při určování lineárních a polynomických závislostí.

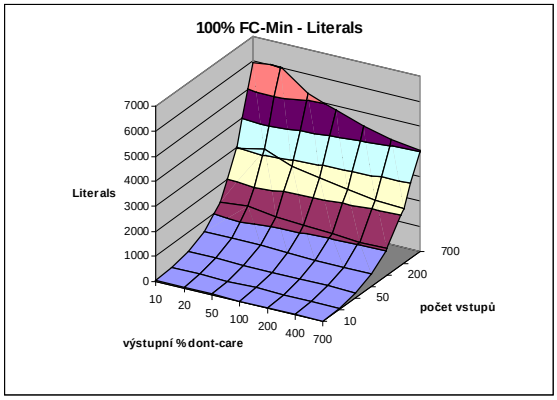
Rozsah počtu vstupních proměnných byl nastaven v sedmi krocích a to 10, 20, 50, 100, 200, 400, 700. Ve stejných krocích jsem volil i počet termů.

4.2.2 Výsledky

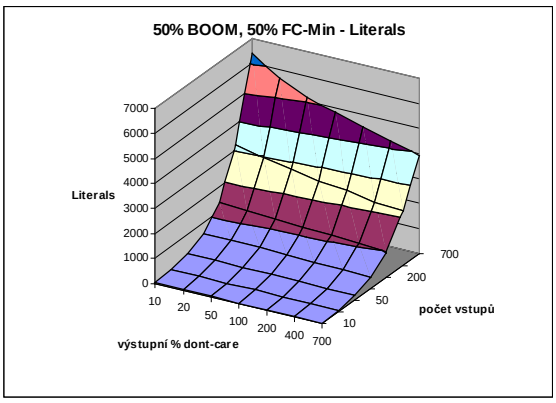
Literals



Obrázek 4.2.1 Test č.1 100% BOOM - Literals



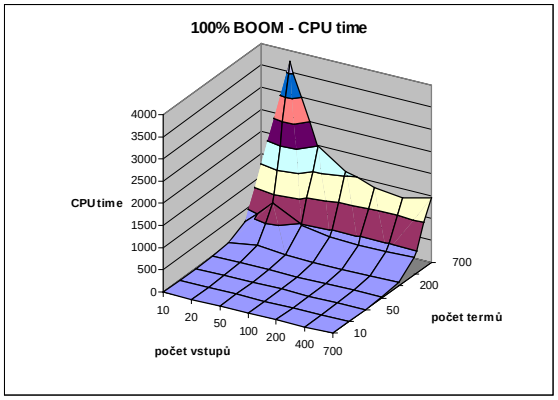
Obrázek 4.2.2 Test č.1 100% FC-Min - Literals



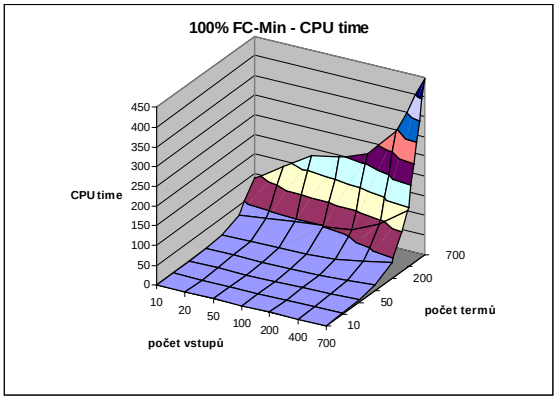
Obrázek 4.2.3 Test č.1 50% BOOM - Literals

V 10ti vstupních proměnných je na tom lépe FC-Min. Potom jsou na tom s BOOMem stejně. A až kolem 700 vstupních proměnných získává mírný náskok BOOM. Test jen pouze potvrdil předpoklady, že BOOM je lepší na mnohem více vstupů. Jen byla zajímavá hranice poměru vstupní, výstupní proměnné, kde se oba minimalizátory sobě vyrovnají.

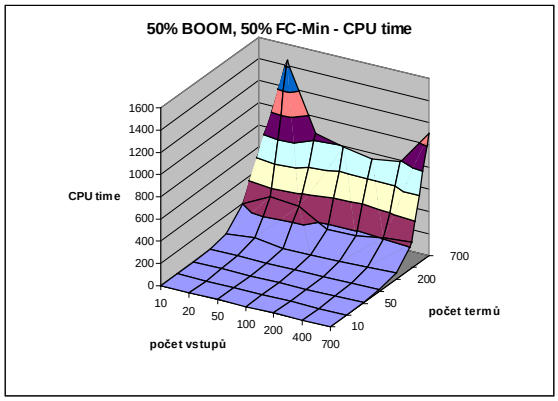
CPU time



Obrázek 4.2.4 Test č.1 100% BOOM - CPU time



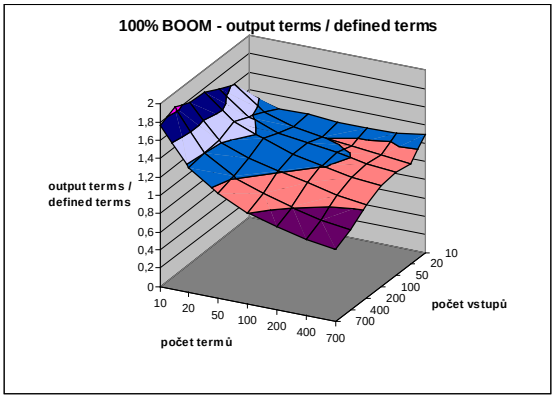
Obrázek 4.2.5 Test č.1 100% FC-Min - CPU time



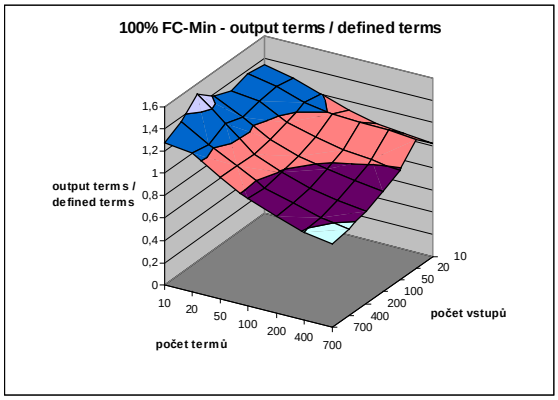
Obrázek 4.2.6 Test č.1 50% BOOM - CPU time

V čase je razantně lepší FC-Min. Při nízkém počtu termů se časy řádově rovnají avšak při počtu termů 700 již FC-Min vede až o jeden řád. Při 700 termech se také ukázala skutečnost kdy BOOM s rostoucím počtem vstupních proměnných zrychluje a FC-Min naopak. Je to samozřejmě způsobeno obráceným postupem obou dvou při hledání implikantů.

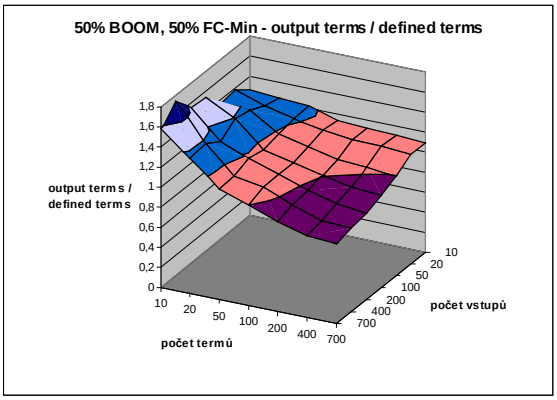
Output terms / defined terms



Obrázek 4.2.7 Test č.1 100% BOOM - terms



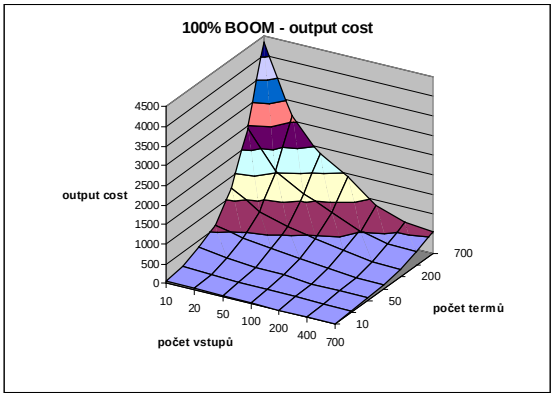
Obrázek 4.2.8 Test č.1 100% FC-Min - terms



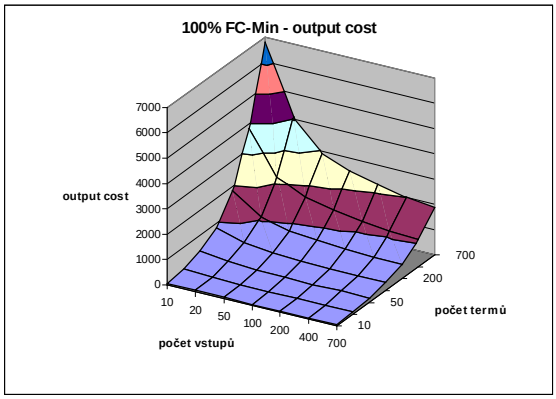
Obrázek 4.2.9 Test č.1 50% BOOM - terms

V počtu termů po minimalizaci vede ve všech případech FC-Min o pár procent před BOOMem. Avšak při počtu vstupů 20 dochází u FC-Minu k lokálnímu maximu. Předpokládám že to bude nějaká statistická chyba, protože nevidím důvod proč by tomu tak melo být.

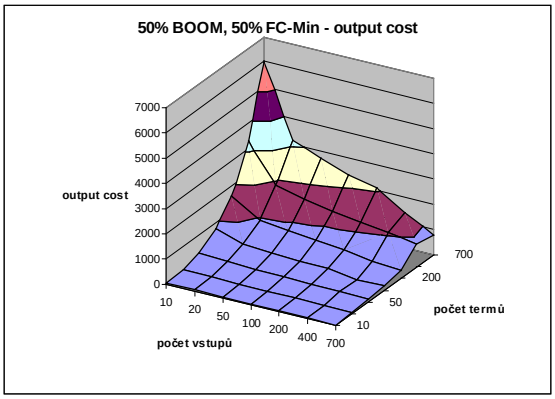
Output cost



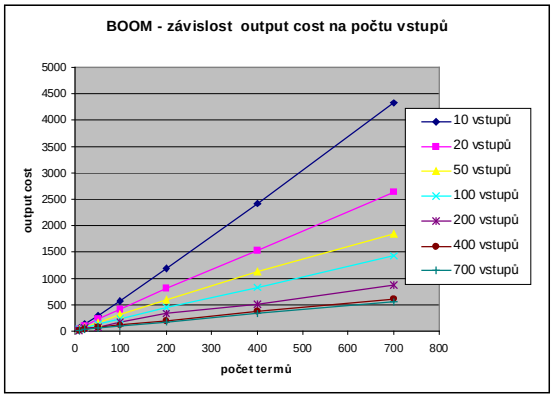
Obrázek 4.2.10 Test č.1 100% BOOM - output cost



Obrázek 4.2.11 Test č.1 100% FC-Min - output cost



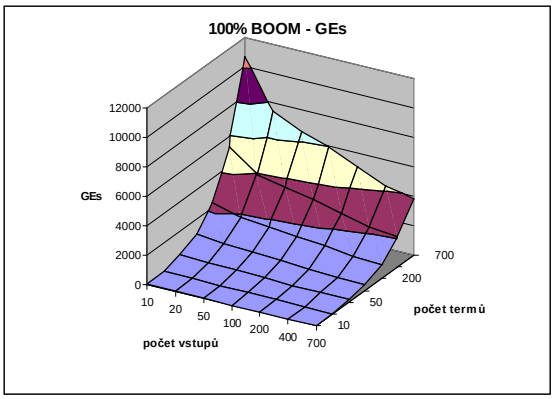
Obrázek 4.2.12 Test č.1 50% BOOM - output cost



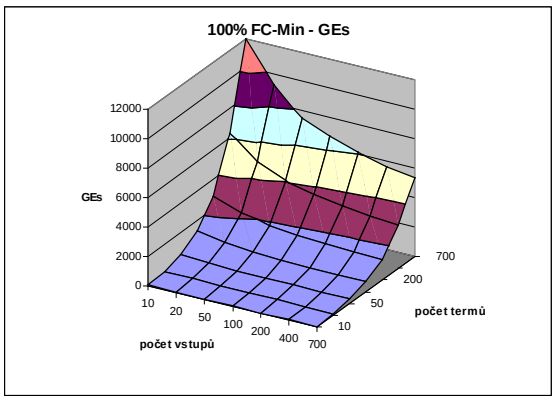
Obrázek 4.2.13 Test č.1 100% BOOM - output cost

Opět se zde potvrzuje že BOOM je výborný na velký počet vstupů a ve ve všech kombinacích termů a vstupů poráží FC-Min. V případě 700 termů a 700 vstupů až o více jak trojnásobek. Jak FC-Min tak BOOM vykazují lineární závislost output cost na počtu termů. Koeficient u lineární funkce je závislý na převrácené hodnotě počtu vstupů.

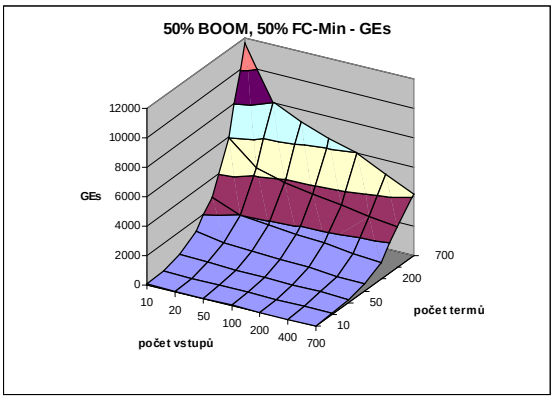
GEs



Obrázek 4.2.14 Test č.1 100% BOOM - GEs



Obrázek 4.2.15 Test č.1 100% FC-Min - GEs



Obrázek 4.2.16 Test č.1 50% BOOM - GEs

Také tyto výsledky nejsou ničím překvapivé. u obou minimalizátorů se projevuje téměř lineární závislost GEs na počtu termů. Ve velké míře dopadl lépe BOOM, avšak FC-Min zde podával srovnatelné výsledky.

4.2.3 Závěr

Tento test v zásadě potvrdil pouze to že na více vstupních proměnných je lepší použít minimalizátor BOOM. Při nastavení kombinace běhu BOOM a FC-Min dochází ke zprůměrování výsledků obou dvou minimalizátorů. Myslím si že při nastavení většího počtu iterací by mohl v tomto případě nést lepší výsledky.

4.3 Test č.2 závislost vstupního procenta don't cares a počtu iterací.

4.3.1 Popis

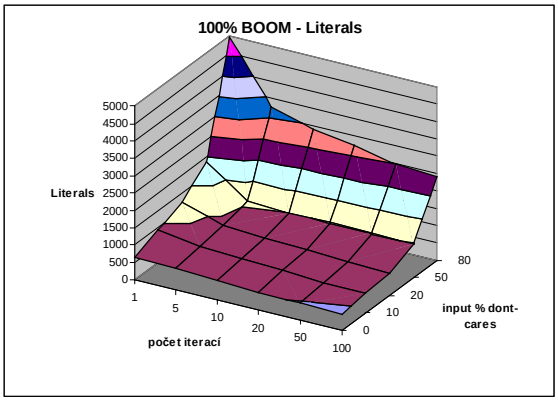
Tímto testem jsem chtěl ukázat že mnou standartně nastavený počet iterací na 10 je dostačující, že výsledná data nebudou výrazně nelineárně zkreslena od výsledků po 100 iteracích.

Trojrozměrné grafy uvedené v tomto testu jsou zkreslené. Poněvadž vynášené hodnoty na osách nereprezentují skutečnou vzdálenost od počátku osy. Je to způsobené omezením softwaru a je si třeba na to dát pozor, hlavně při určování polynomických závislostí.

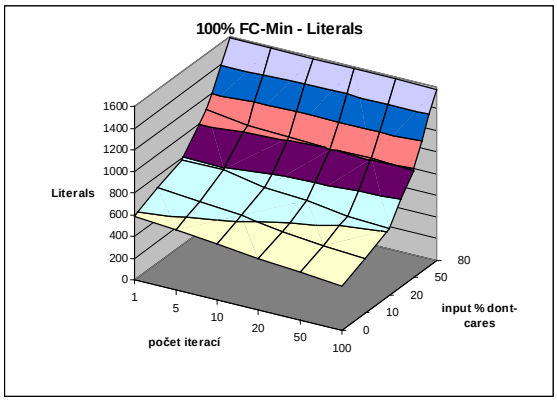
Počet iterací byl nastaven podle "pseudo" logaritmické řady na vzorky 1, 5, 10, 20, 50, 100. Druhá proměnná bylo vstupní procento don't care a to v tomto rozsahu 0, 10, 20, 50, 80. Počet vstupů byl nastaven na 100, výstupů 20, termů 100.

4.3.2 Výsledky

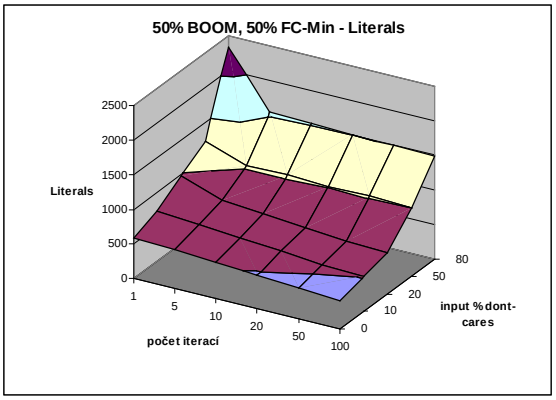
Literals



Obrázek 4.3.1 Test č.2 100% BOOM - Literals



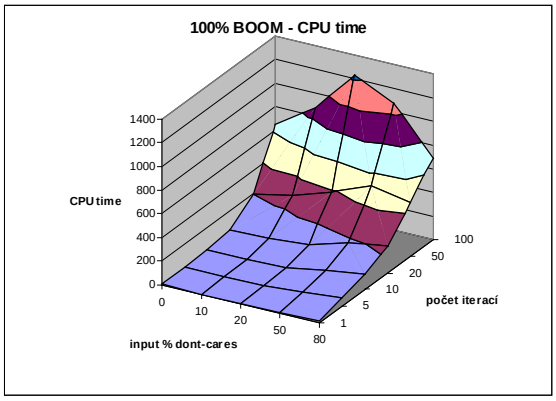
Obrázek 4.3.2 Test č.2 100% FC-Min - Literals



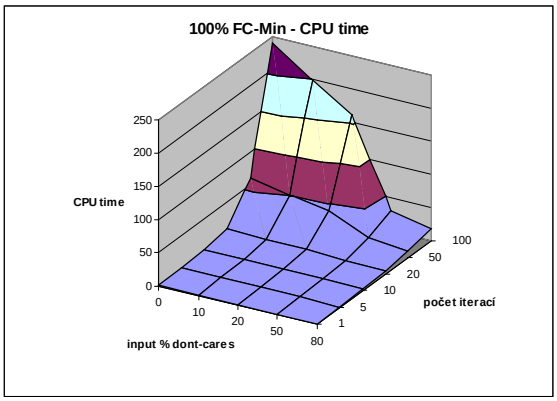
Obrázek 4.3.3 Test č.2 50% BOOM - Literals

V případě FC-Minu i BOOMu je závislost při nulovém procentu vstupních don't care v testovaném rozsahu téměř lineární. S rostoucím počtem iterací klesá počet literálů. Oba dva vykazují skoro tytéž výsledky. Avšak u FC-Minu se postupně závislost na iteracích snižuje s rostoucím počtem don't care na vstupu. při 80% již nedochází k ovlivnění výsledku počtem iterací. U BOOMu je tomu přesně naopak, původně téměř linearní pokles, se počtem vstupních procent don't care mění silně konvexní klesání literálů v závislosti na počtu iterací. V těchto vyšších procentech vstupních don't care je lepší minimalizátor FC-Min který je na tom lépe 2krát až 4krát. Zajímavé bylo že kombinace obou dvou vyšla skoro ve všech případech o trochu lépe než samotný FC-min.

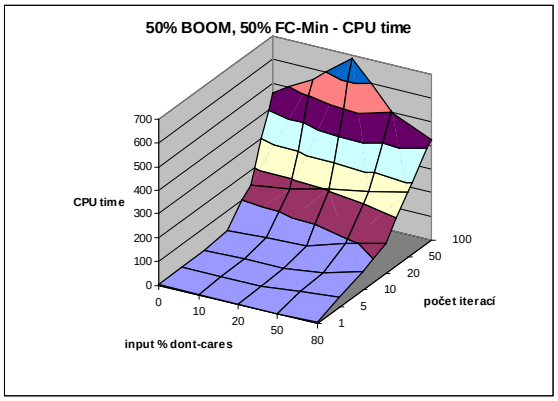
CPU time



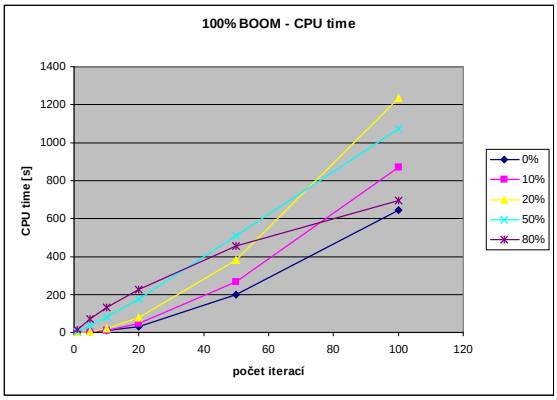
Obrázek 4.3.4 Test č.2 100% BOOM - CPU time



Obrázek 4.3.5 Test č.2 100% FC-Min - CPU time



Obrázek 4.3.6 Test č.2 50% BOOM - CPU time

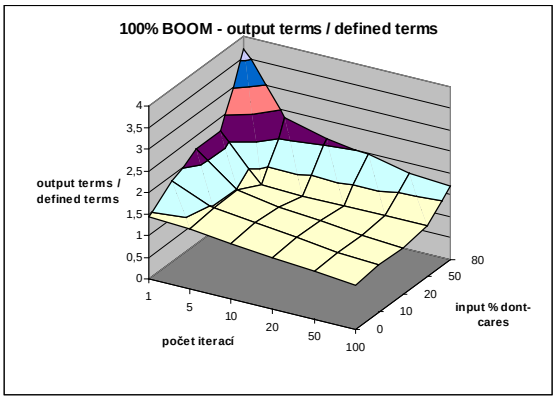


Obrázek 4.3.7 Test č.2 100% BOOM - CPU time

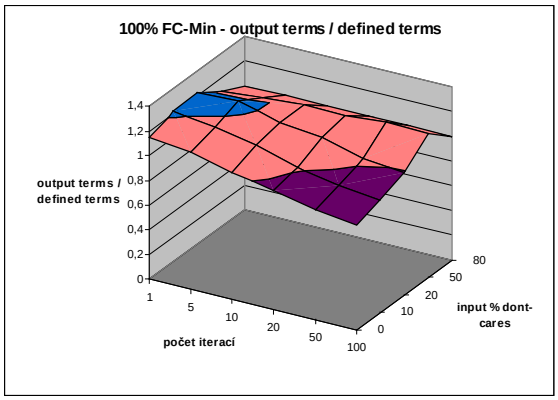
Celkově se dá říci že v čase byl lepší FC-Min. Zde ovšem byla výraznější tendence růstu času v závislosti na iteracích, nežli u BOOMu, který ovšem vykazoval mnohem horší časy. Zajímavé bylo pozorovat změnu obou minimalizátorů při rostoucím procentu vstupní don't care. Křivka závislost v BOOMu se zvyšovala ale v 50% se rostoucí konvexní funkce změnila na lineární a ve vyšších procentech dokonce na konkávní viz Obrázek 4.3.7 Kdyby tento trend pokračoval i pro více iterací, pak by to znamenalo že čas limtuje k nějaké hodnotě. Podle mne spíše dochází ke snížení počtu generovaných dříve nenalezených přímých implikantů v každé iteraci, takže by průběh měl mít dále lineární charakter.

FC-Min vykazoval zmenšování koeficientů u pravděpodobně polynomické křivky závislosti času na počtu iterací s rostoucím počtem procent vstupních don't care.

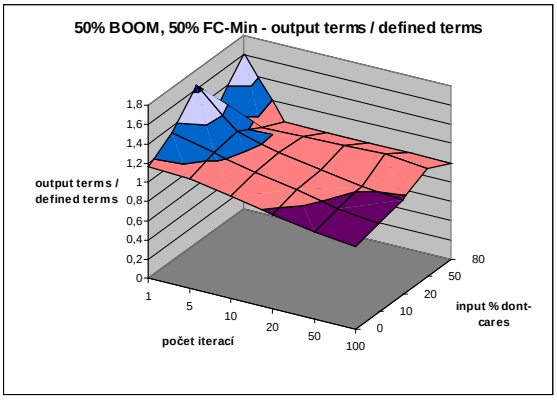
Output terms / defined terms



Obrázek 4.3.8 Test č.2 100% BOOM - terms



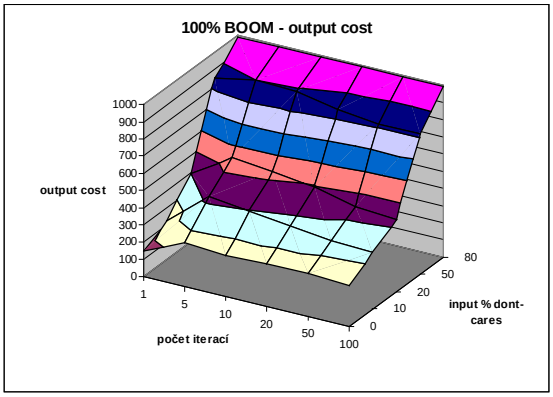
Obrázek 4.3.9 Test č.2 100% FC-Min - terms



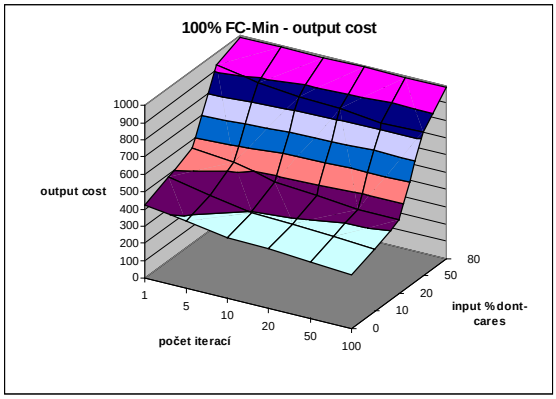
Obrázek 4.3.10 Test č.2 50% BOOM - terms

Zde se ukazuje pěkný rozdíl mezi BOOMem a FC-Minem. FC-Min již v první iteraci udělá relativně správný výsledek, který už jen po malých krůčkách zlepšuje. Zajímavé ještě je že v závislosti na procentu vstupních don't care vykazuje nejhorší výsledky kolem 50% poté se již zase zlepšuje. BOOM naopak v první iteraci udělá hodně špatný výsledek který dále razantně zlepšuje v nejbližších iteracích. Až výsledek po páté iteraci se dá relativně srovnávat s výsledkem po 100 iteracích. Také se výsledek zhoršuje se zvyšujícím se procentem vstupních don't care. Úspěšnost FC-Minu si vysvětluji lepším nalezením skupinových implikantů už při první iteraci.

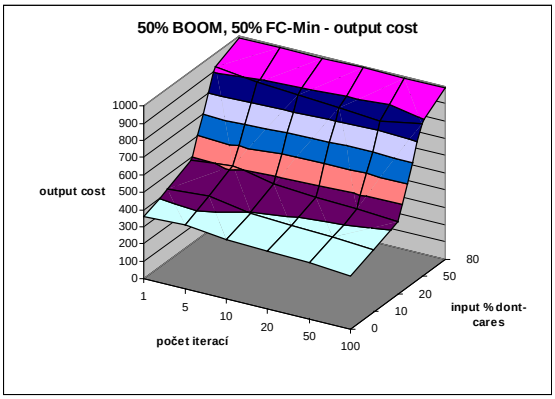
Output cost



Obrázek 4.3.11 Test č.2 100% BOOM - output cost



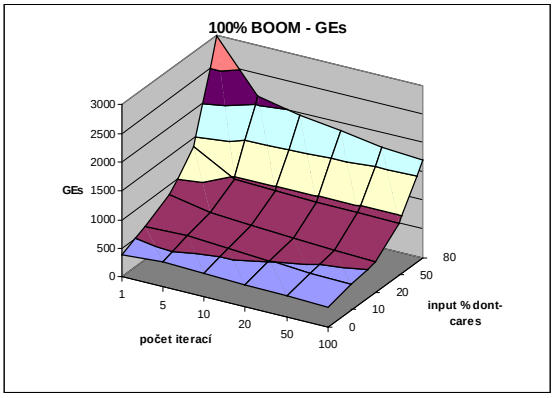
Obrázek 4.3.12 Test č.2 100% FC-Min - output cost



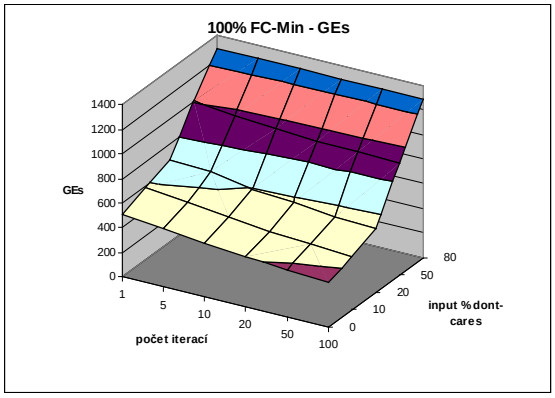
Obrázek 4.3.13 Test č.2 50% BOOM - output cost

Je zajímavé, že oba minimalizátory při 80% vstupních don't care měli totožné výsledky a nebyly ovlivněné počtem iterací. Což si vysvětluji tím že už v první iteraci našli oba dva nejlepší řešení, poněvadž bylo mnohem snažší hledat přímé implikanty. Celkově byl na tom lépe BOOM a dokonce první iterací udělal lepší výsledky než po 100 iteracích. Je to způsobeno pravděpodobně tím že je standartně nastaveno aby se minimalizovalo output cost + literals.

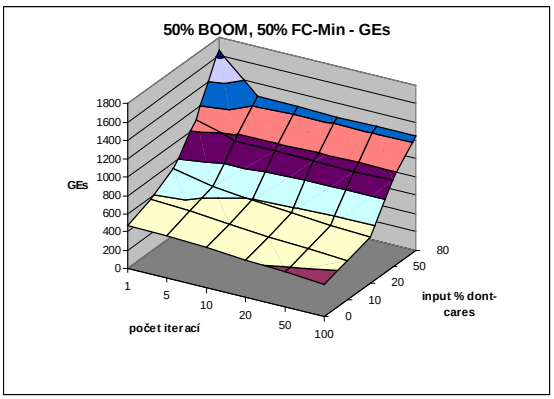
GEs



Obrázek 4.3.14 Test č.2 100% BOOM - GEs



Obrázek 4.3.15 Test č.2 100% FC-Min - GEs



Obrázek 4.3.16 Test č.2 50% BOOM - GEs

Zde byly oba minimalizátory na stejno při žádných vstupních don't cares. Při zvyšování si lépe vedl FC-Min, až se dostal na konstantní hodnotu nezávislou na iteracích při 80% vstupních don't care, která byla přibližně 2 krát lepší nežli u BOOMu. BOOM se choval se stejnými tendencemi jako u literálů. První iterací získal velmi špatný výsledek, který ovšem v pár nejbližších iteracích srazil na únosnou mez oproti 100 iteracím. Při spolupráci obou dvou došlo k průměrování výsledků

4.3.3 Závěr

V tomto testu se ukázalo že opravdu 10 iterací dává v drtivé většině relevantní výsledek oproti 100 iteracím. Během fáze z 10ti do 100 iterací se výsledek zlepší u BOOMu v průměru asi o 10%. FC-Min je ještě na počet iterací odolnější a v některých případech dává velmi slušné výsledky již v první iteraci. Časově je mnohem rychlejší za daných podmínek FC-Min. Kombinace obou dvou minimalizátorů jednou překvapila že dala výsledek lepší než samotné minimalizátory. Jinak dávala statisticky průměrné výsledky obou dvou.

4.4 Test č.3 závislost výstupního procenta don't care a počtu výstupů

4.4.1 Popis

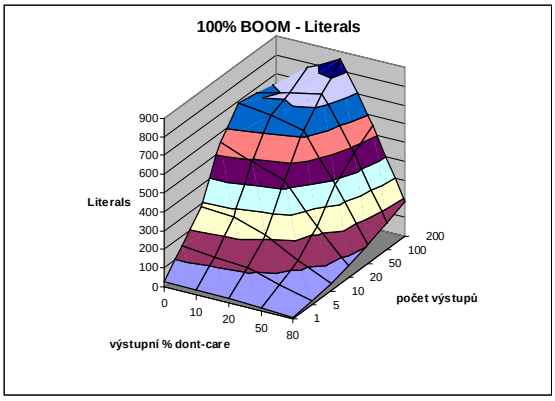
Tento test byl určen na testování závislostí při změně výstupů. Protože FC-Min začíná minimalizovat od výstupů, myslel jsem že by se mohly ukázat zajímavé skutečnosti.

Trojrozměrné grafy uvedené v tomto testu jsou zkreslené. Poněvadž vynášené hodnoty na osách nereprezentují skutečnou vzdálenost od počátku osy. Je to způsobené omezením softwaru a je si třeba na to dát pozor, hlavně při určování polynomických závislostí.

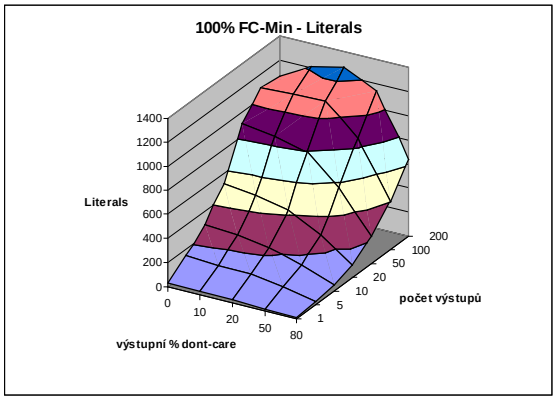
Počet výstupů byl nastaven na hodnoty 1, 5, 10, 20, 50, 100,200. Druhá proměnná bylo výstupní procento don't care a to v tomto rozsahu 0, 10, 20, 50, 80. Počet vstupů byl nastaven na 100, termů 100.

4.4.2 Výsledky

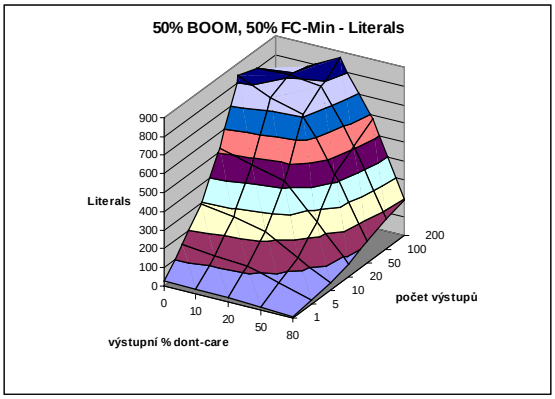
Literals



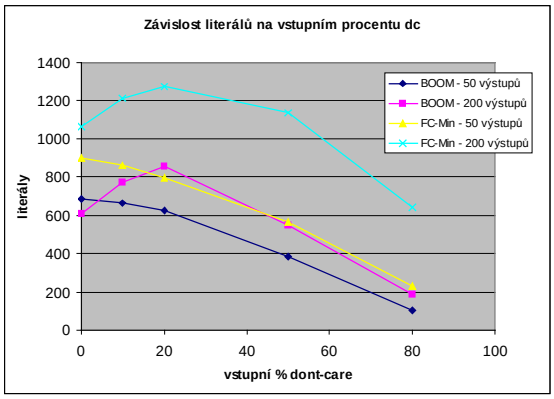
Obrázek 4.4.1 Test č.3 100% BOOM - Literals



Obrázek 4.4.2 Test č.3 100% FC-Min - Literals



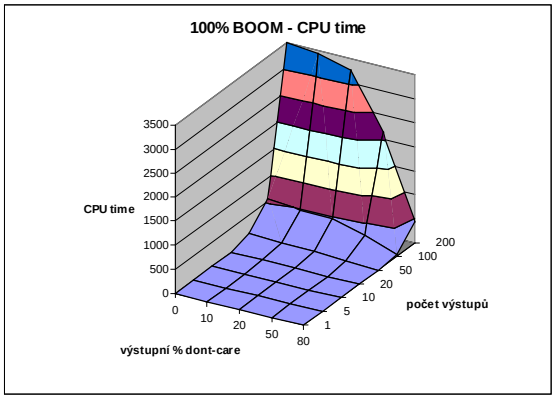
Obrázek 4.4.3 Test č.3 50% BOOM - Literals



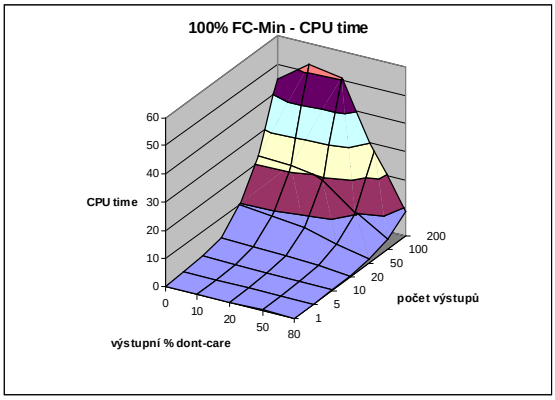
Obrázek 4.4.4 Test č.3 Srovnání - Literals

Zajímavé že nejvíce literálů je pro 20% výstupních don't care viz Obrázek 4.4.4. Tento jev si neumím rozumně vysvětlit. Možná je to způsobeno volbou malého počtu iterací, avšak při získávání dat v okolí tohoto jevu. tzn. Při 0%-20% input don't cares a 200 výstupů BOOM i FC-Min běžel poměrně dlouhou dobu a vznikalo již dříve zmíněné swapování. Pro 80% výstupních don't care literály ve zkoumaném rozsahu stoupaly lineárně spolu s počtem výstupů. V tomto případě byl na tom minimalizátor BOOM lépe ve všech situacích a to až o 80%. Kombinace obou minimalizátorů si vzala maximum z BOOMu poněvadž dopadla s ním téměř identicky.

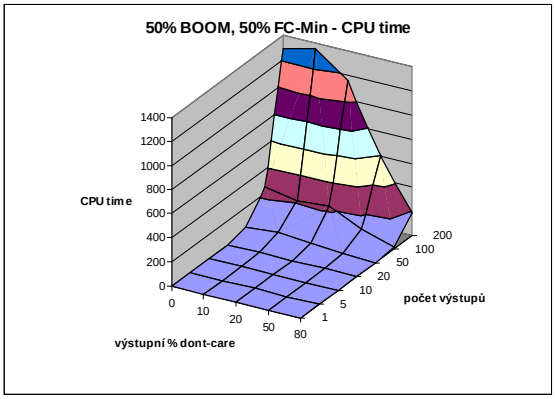
CPU time



Obrázek 4.4.5 Test č.3 100% BOOM - CPU time



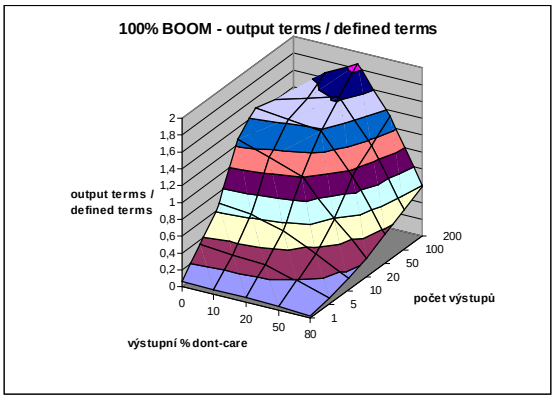
Obrázek 4.4.6 Test č.3 100% FC-Min - CPU time



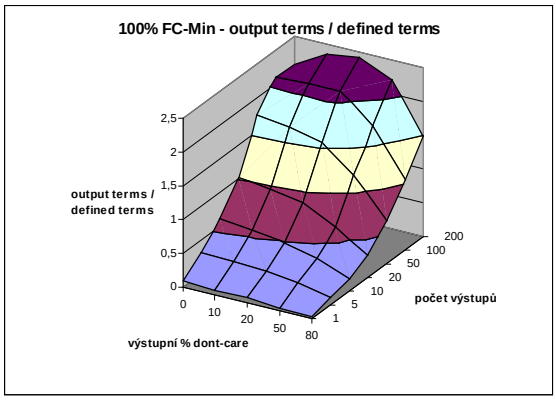
Obrázek 4.4.7 Test č.3 50% BOOM - CPU time

Zde se žádné překvapivé výsledky neodehraly. Podle očekávání časová náročnost u BOOMu rostla kvadratickou řadou v závislosti na počtu výstupech. Je to způsobeno vygenerováním velkého počtu implikantů, z kterých se potom sekvenčně snažíme rozšířit na skupinové. Časová složitost FC-Min rostla lineárně. BOOM byl jen o trochu rychlejší v při nastavení jednoho výstupu a občas pěti. Jinak v rychlosti kraluje FC-Min. Zvyšující se výstupní procento don't care jen snižovalo koeficient u křivek zavislosti na počtu výstupech. Bylo to pravděpodobně způsobeno tvorbou větších přímých implikantů a tím i snížení jejich počtu a menší časovou náročností.

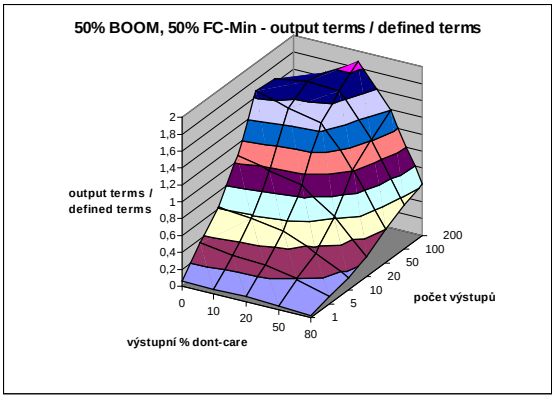
Output terms / defined terms



Obrázek 4.4.8 Test č.3 100% BOOM - terms



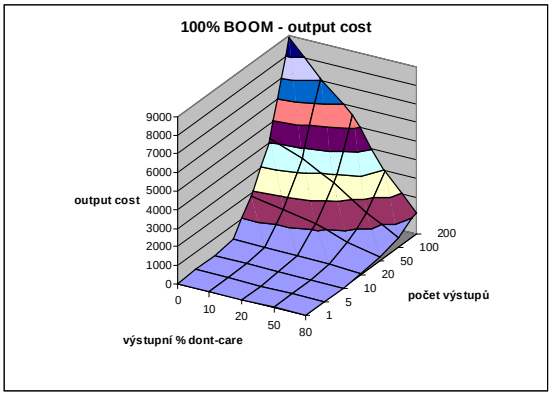
Obrázek 4.4.9 Test č.3 100% FC-Min - terms



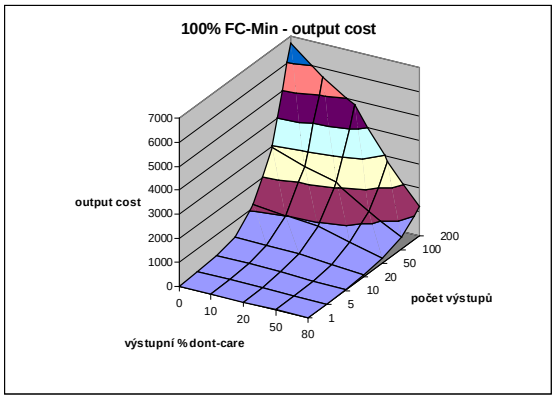
Obrázek 4.4.10 Test č.3 50% BOOM - terms

Tyto grafy jako by z oka vypadly kritériu literálů. Nedokážu si vysvětlit čím to je že, protože v jiných testech vycházely literály a termy rozdílně. Rozhodně platí to samé co jsem již psal výše. Lépe dopadl minimalizátor BOOM a v některých situacích je lepší než FC-Min až o 80%.

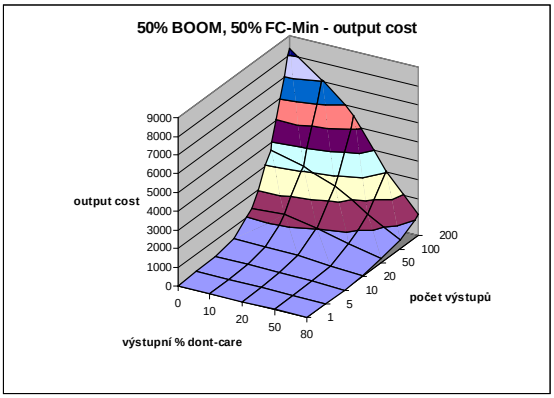
Output cost



Obrázek 4.4.11 Test č.3 100% BOOM - output cost



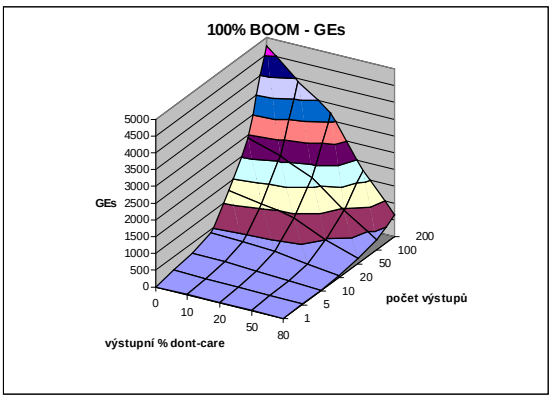
Obrázek 4.4.12 Test č.3 100% FC-Min - output cost



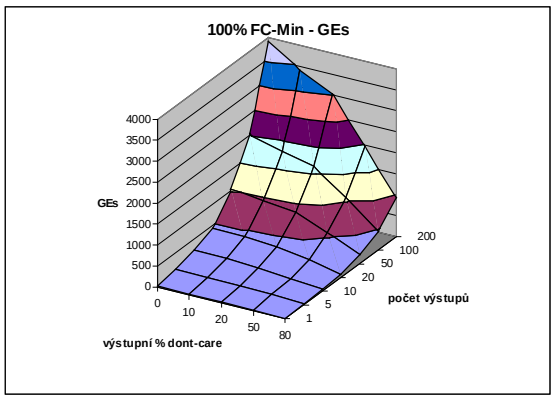
Obrázek 4.4.13 Test č.3 50% BOOM - output cost

V případech přibližně do 20 výstupních termů dává lepší výsledky BOOM. V ostaních případech je tomu naopak, takže kvalitnější výsledky dává FC-Min. Což není vůbec překvapující. Kombinace obou minimalizatérů, funguje spíše jako BOOM, ale jsou zde místa kde naopak kopíruje FC-Min. Dá se říct, že spíše se přiklání vždy k horší variantě. Takže bych kombinaci nedoporučoval použít.

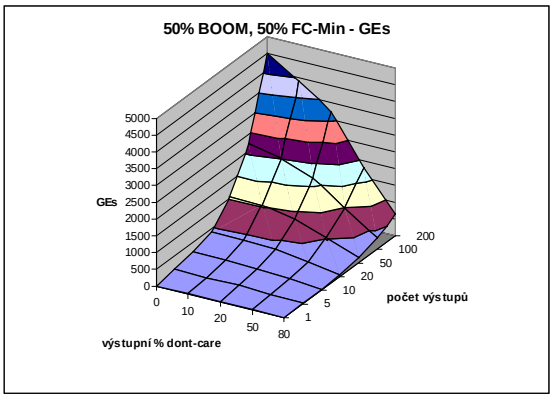
GEs



Obrázek 4.4.14 Test č.3 100% BOOM - GEs



Obrázek 4.4.15 Test č.3 100% FC-Min - GEs



Obrázek 4.4.16 Test č.3 50% BOOM - GEs

V Případě GEs lze říci že pokud je výstupní procento don't care nad 50% dává BOOM lepší výsledky nežli FC-Min. Je to pravděpodobně způsobeno tím že přímé implikanty genrované BOOMem lze pa díky výstupním don't cares jednodušeji rozšířit na skupinové. FC-Min nalezne velké pokrytí výstupu pomocí don't cares, ke kterým však pak h§ře nachází skupinové implikanty. Kombinace se spíše přiklonila na stranu BOOMu, je to logické, protože vygeneroval pravděpodobně kvalitnější skupinové implikanty.

4.4.3 Závěr

Tento test opravdu nepřinesl nic nového. Předpokládal jsem že počet výstupů v kontextu s procentem výstupních don't care bude mít výraznější vliv na FC-Min avšak ukázalo se, že ve velké míře byl lepší minimalizatér BOOM mimo kritéria output cost a GEs, kde bych doporučil při vyšším počtu výstupu než 20 použít FC-Min. FC-Min je obecně také mnohem časově méně náročnější protože má přibližně linearní časovou složitost v závislosti na počtu výstupů.

4.5 Test č.4 závislost vstupního poměru 1:0 a výstupního poměru 1:0

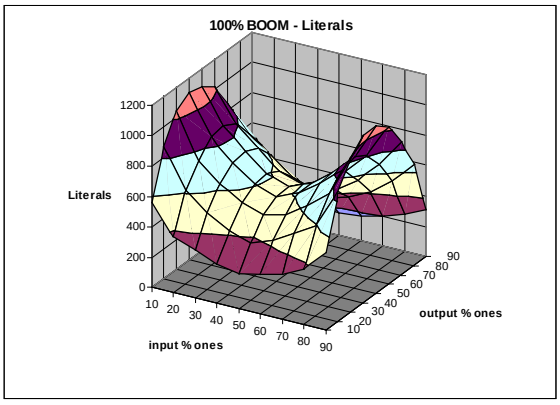
4.5.1 Popis

Tento test byl ze všech asi nejzajímavější poněvadž testoval něco nového, co ještě nebylo minimalizátory testováno. Tento test byl především určen na porovnání chování obou minimalizátorů v závisosti na různém poměru jedniček a nul v termech jak ve vstupech tak i ve výstupech.

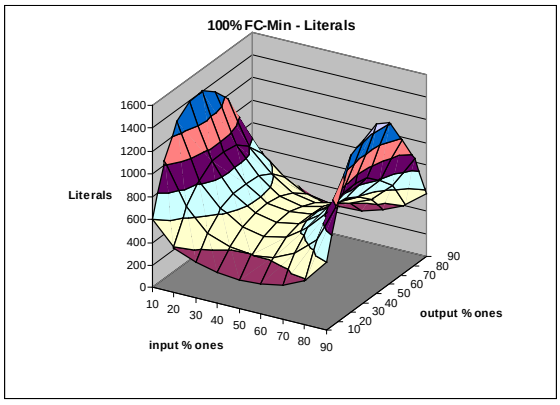
Rozsah vstupního poměru byl od 10% do 90% v korku po 10%, rozsah výstupního poměru byl taktéž od 10% do 90% v kroku po 10%. pocet vstunpích proměnných byl nastaven na 100, výstupních na 20 a počet termů na 100. Jako ve všech testech i tento byl test vyhodnocován na všechny výstupní kritéria.

4.5.2 Výsledky

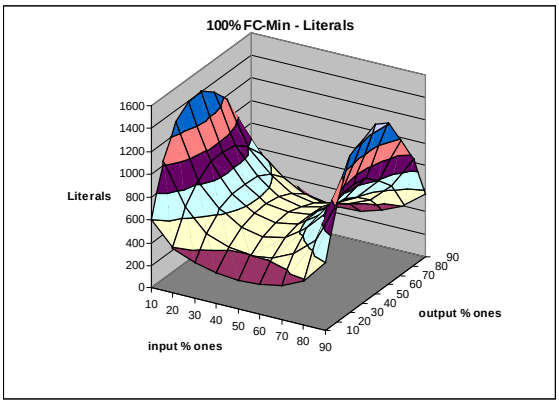
Literals



Obrázek 4.5.1 Test č.4 100% BOOM - Literals



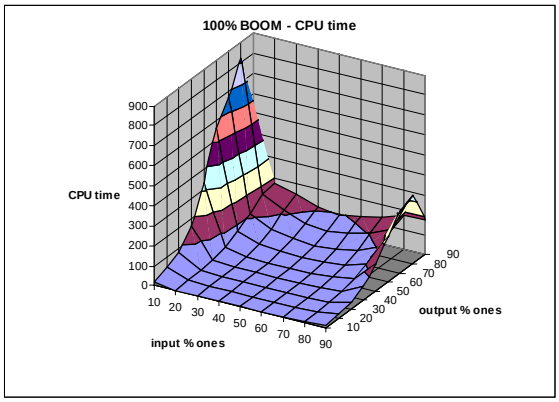
Obrázek 4.5.2 Test č.4 100% FC-Min - Literals



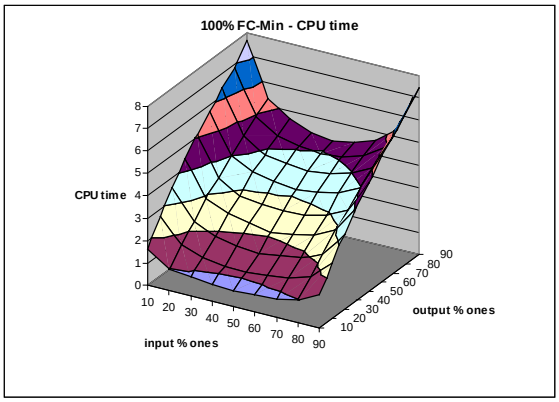
Obrázek 4.5.3 Test č.4 50% BOOM - Literals

Z grafů je patrné že z hlediska literálů v případech vstupů s velkým nepoměrem nul a jedniček byl lepší minimalizátor BOOM a to v případě vyváženého výstupu o až 1,5krát. Dále je z výsledků zajímavé že BOOM byl v případě většího poměru jedniček na výstupu (60+%) lepší než v případě malého počtu jedniček na výstupu. Nabízí se zlepšení celého algoritmu, tím že při zjištění většího počtu jedniček než nul na výstupu. V celém výstupu prohodíme jedničky za nuly a naopak. V případě přibližně stejného počtu nul a jedniček byl z hlediska litterálů o pár procent lepší algoritmus FC-Min.

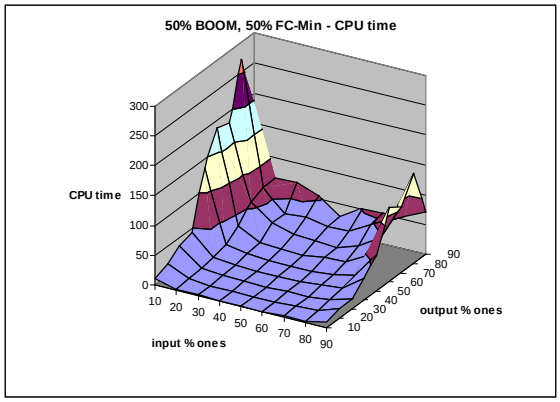
CPU time



Obrázek 4.5.4 Test č.4 100% BOOM - CPU time



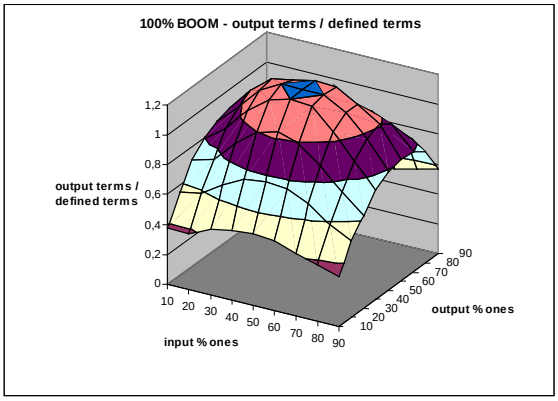
Obrázek 4.5.5 Test č.4 100% FC-Min - CPU time



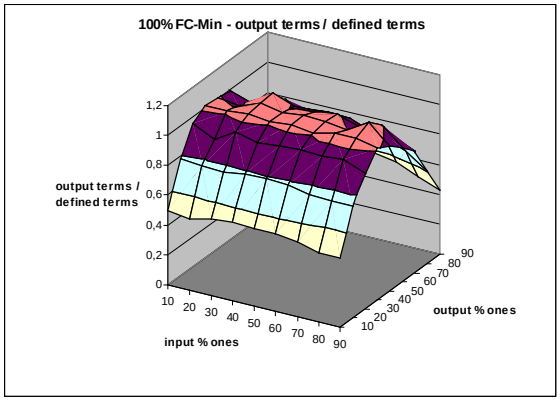
Obrázek 4.5.6 Test č.4 50% BOOM - CPU time

Ohledně spotřebovaného času na minimalizaci jasně vede minimalizátor FC-Min. V nejmenším rozdílu je asi 2x rychlejší a to v rozvnoměrném rozložení jedniček na vstupu a co nejméně jedniček na výstupu. V krajních mezích FC-Min vítězil až o 2,5 řádu. V nízkém počtu jedniček na vstupu i výstupu navíc BOOM vykazoval velký rozptyl času. tzn. velmi záleželo na vygenerované funkci.

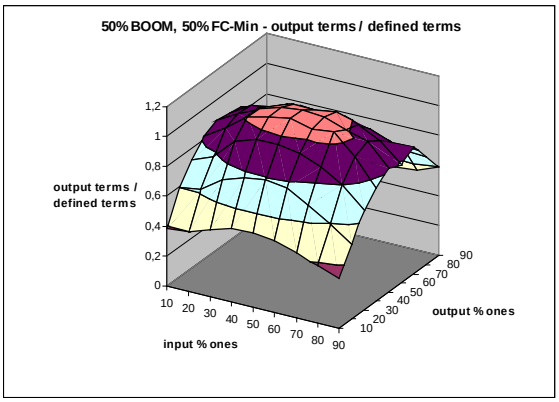
Output terms / defined terms



Obrázek 4.5.7 Test č.4 100% BOOM - terms



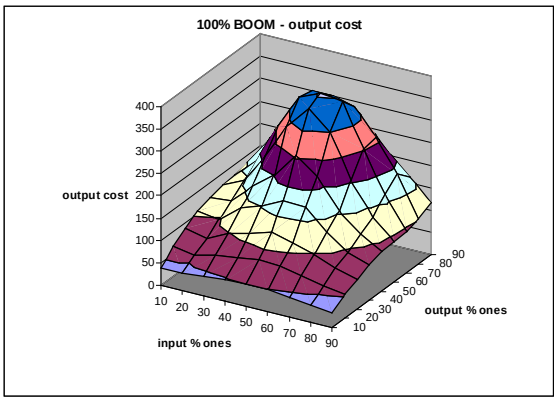
Obrázek 4.5.8 Test č.4 100% FC-Min - terms



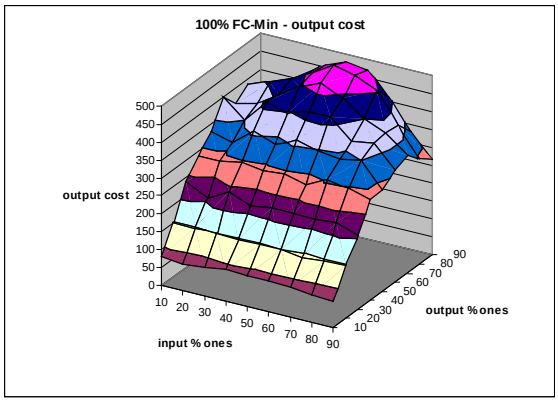
Obrázek 4.5.9 Test č.4 50% BOOM - terms

Tyto grafy vycházejí poměrně shodně. Zajímavostí je, že FC-Min není vůbec citlivý na změnu počtu jedniček na vstupu, což si vysvětluji jeho obráceným postupem při tvorbě implikantů. Ve velké většině je také z tohoto hlediska lepší nebo alespoň stejně dobrý nežli BOOM. BOOM získává náskok pouze při vstupu 10% jedniček respektivě 90% při výstupu mezi 30% až 70% jedniček. Minimalizátor běžící v poměru 50% BOOM a 50% FC-Min vykazuje velmi dobrou aproximaci mezi oběma výsledky.

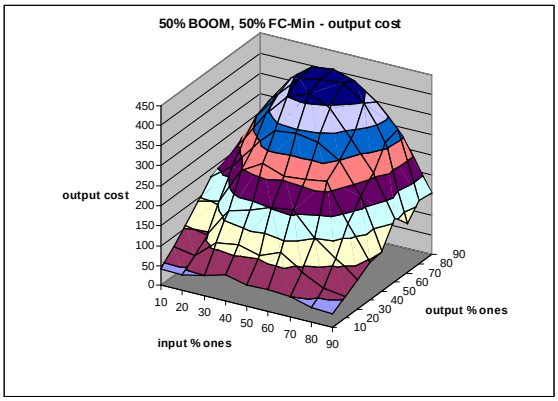
Output cost



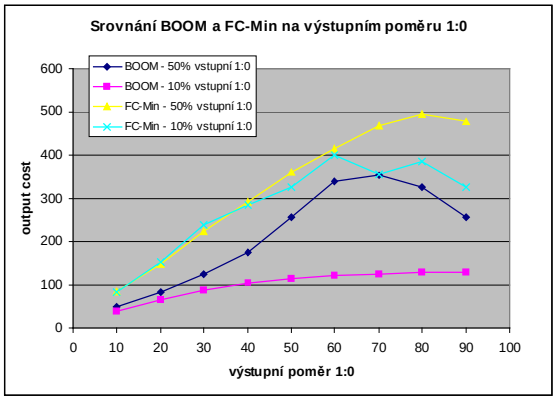
Obrázek 4.5.10 Test č.4 100% BOOM - output cost



Obrázek 4.5.11 Test č.4 100% FC-Min - output cost

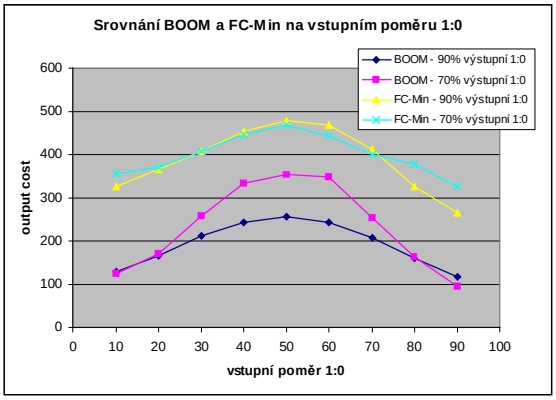


Obrázek 4.5.12 Test č.4 50% BOOM - output cost



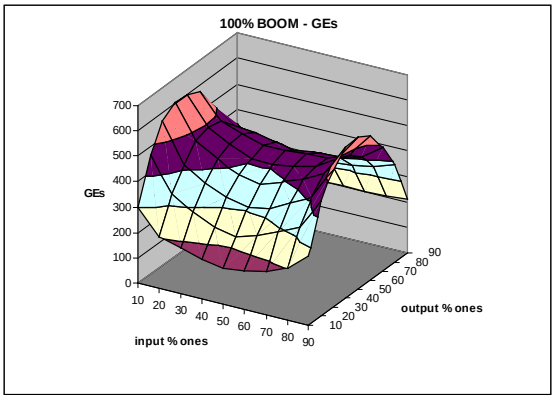
Obrázek 4.5.13 Test č.4 Srovnání - output cost

Tyto grafy považuji za nejvydařenější z celého testu č.4. Oba dva minimalizátory měli maximum, když vstup byl rovnoměrně vyvážen a výstup měl 70% jedniček. To je vzhledem ke kritéiu logické. Zatímco však BOOM se k vrcholu přibližoval pozvolněji spíše konvexně, tak FC-Min přesně naopak s tendencí dosti konkávní. Zase bych si to vysvětlil rozdílným způsobem tvoření skupinových implikantů. V celém rozsahu BOOM překonává FC-Min. Má menší output cost mezi 33% až 80% vztaženo k výsledkům FC-Minu. Kombinace obou minimalizátorů, věrně2 průměruje jejich výsledky.

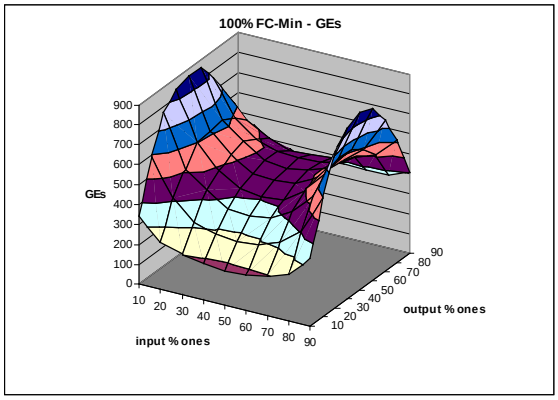


Obrázek 4.5.14 Test č.4 Srovnání - output cost

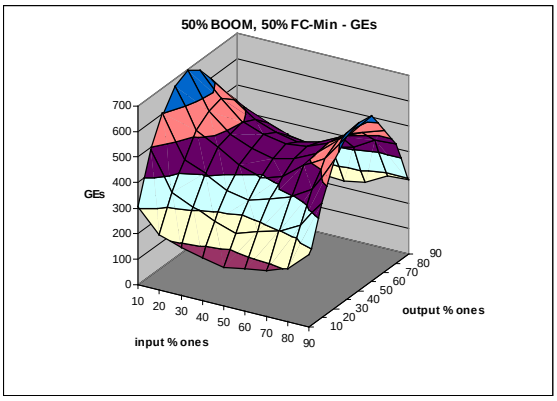
GEs



Obrázek 4.5.15 Test č.4 100% BOOM - GEs



Obrázek 4.5.16 Test č.4 100% FC-Min - GEs



Obrázek 4.5.17 Test č.4 50% BOOM - GEs

I zde podobně jako v kritériu literálů v okrajových podmínkách vstupů byl BOOM výrazně lepší nežli FC-Min. Zajímavé také je že BOOM při velkém počtu jedniček na výstupu (90%) již nebyla závislost výstupu na počtu jedniček na vstupu. Další zajímavostí je, že podobně jako BOOM u literálů, zde FC-Min vykazoval jistou odchylku v případě většího poměru jedniček na výstupu (60+%). FC-Min byl v tomto případě horší nežli v případě malého počtu jedniček na výstupu. I toto nabízí jisté zamyšlení nad využitím této skutečnosti.

Ještě je zajímavé že pouze u tohoto testu se grafy GEs a output cost kategoricky lišily.

4.5.3 Závěr

Skoro ve všech případech byl zde lepší algoritmus BOOM nežli FC-Min, což je ovšem zaplaceno velkou časovou náročností. Bylo to pravděpodobně způsobeno nastavením počtu vstupů a výstupů. Myslím že se v tomto ukázala velmi zajímavá věc a to rozdílné chování minimalizátorů na různé poměry jedniček především na výstupu. Tato problematika by si zajisté žádala další důkladnější testování. A případné využití zjištěných závislostí na zvýšení efektivity těchto minimalizátorů. Takže pokud hledáme minimalní počet hradel a nezáleží na čase, volil bych jednoznačně BOOM, v případě rychlosti je mnohem výkonější minimalizátor FC-Min. Rozumný kompromis mezi oběma variantama nabízí nastavení kombinace obou dvou.

4.6 Test č.5 závislost vstupní granularity don't cares průměrným poměrem 50% a závislost výstupní granularity 1:0 s průměrným poměrem 50%

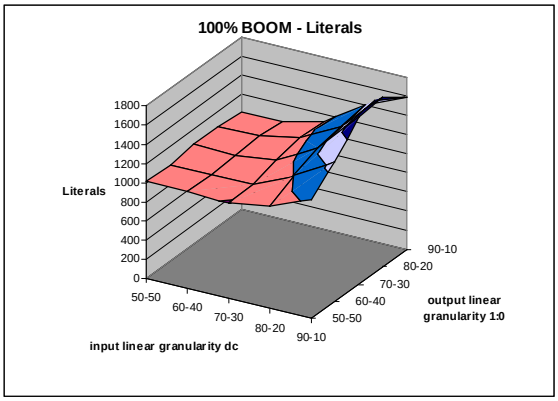
4.6.1 Popis

V tomto testu jsem chtěl otestovat další už poměrně složitěji hledatelné rozdíly ve vstupních tabulkách.

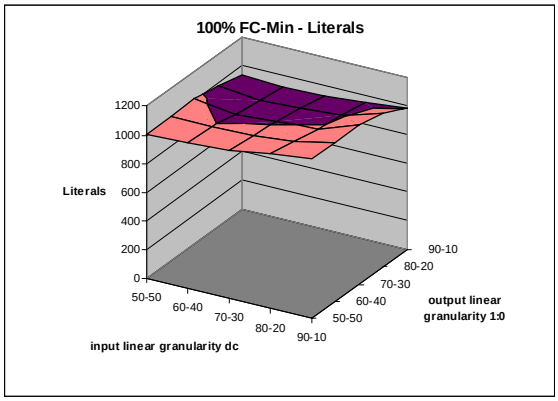
Rozsah vstupní granularity don't care byl nastaven v pěti krocích a v každém byla průměrná granularita 50%. Kroky teda jsou: 90-10, 80-20, 70-30, 60-40 a 50-50. například granularita 1:0 80-35 znemená že na prvním termu bude 80% jedniček lineárně klesající až do posledního termu kde bude jen 35% jeniček. Taktéž byla zvolena granularita 1:0 na výstupu

4.6.2 Výsledky

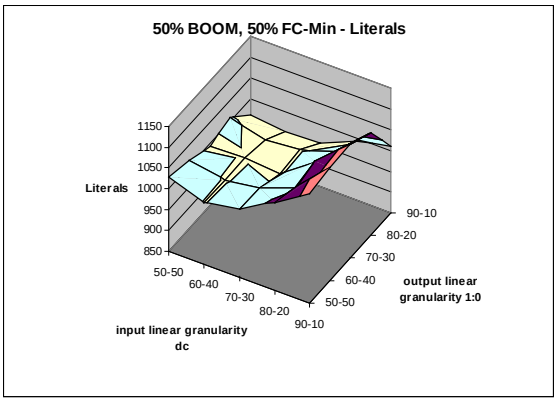
Literals



Obrázek 4.6.1 Test č.5 100% BOOM - Literals



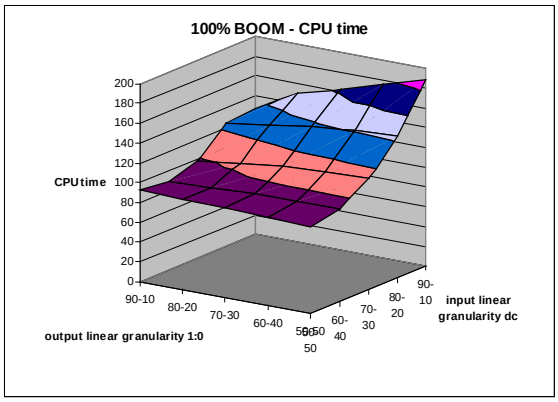
Obrázek 4.6.2 Test č.5 100% FC-Min - Literals



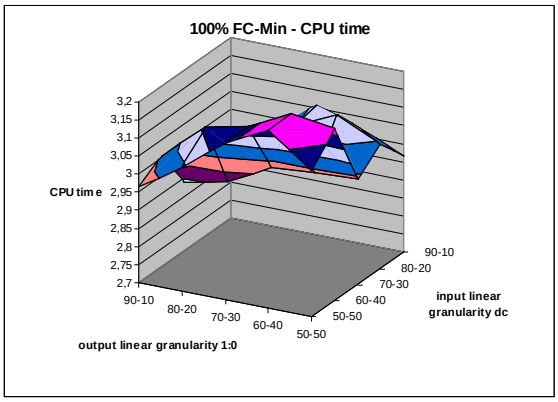
Obrázek 4.6.3 Test č.5 50% BOOM - Literals

Je videt že v případě literálů nemá ani jedna granularita vliv na minimalizátor FC-Min. BOOM je ovlivněn vstupní granulairtou don't care, při zvyšující se nerovnoměrném rozložení jedniček na vstupu. Zhoršuje výsledky až o 80%.

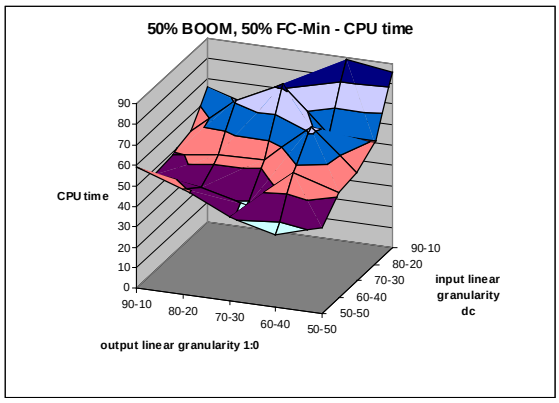
CPU time



Obrázek 4.6.4 Test č.5 100% BOOM - CPU time



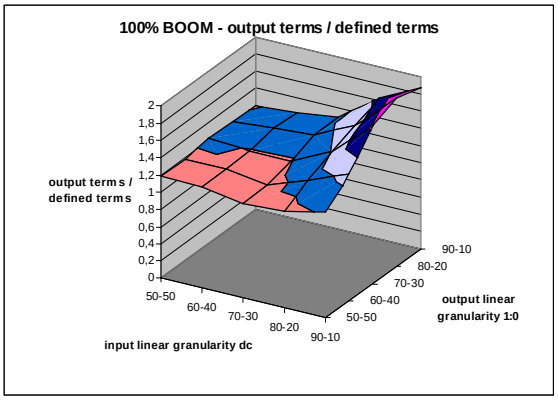
Obrázek 4.6.5 Test č.5 100% FC-Min - CPU time



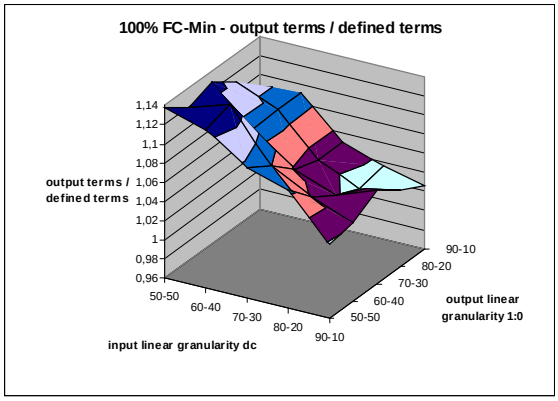
Obrázek 4.6.6 Test č.5 50% BOOM - CPU time

V čase opět je výrazně lepší FC-Min a to až o dva řády. FC-Min byl velmi nevyrovnaný ve výsledcích a vykazoval náhodné odchylky spíše než závislosti na proměnných. Bylo to pravděpodobně velmi krátkým časem který minimalizátor běžel. Zatímco BOOM vykazoval závislost a to, že při velké granularitě don't care na vstupech pracuje až 2krát pomaleji. Částečně to zlepšuje vysoká granularita 1:0 na výstupu.

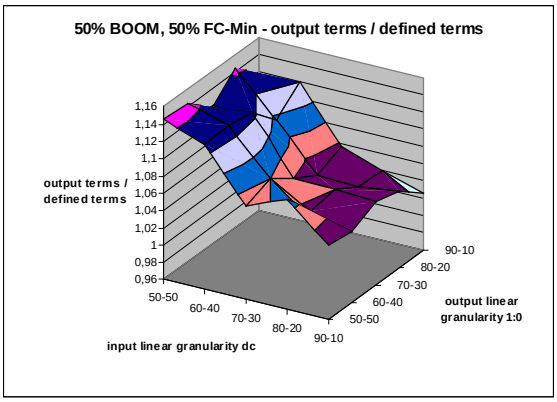
Output terms / defined terms



Obrázek 4.6.7 Test č.5 100% BOOM - terms



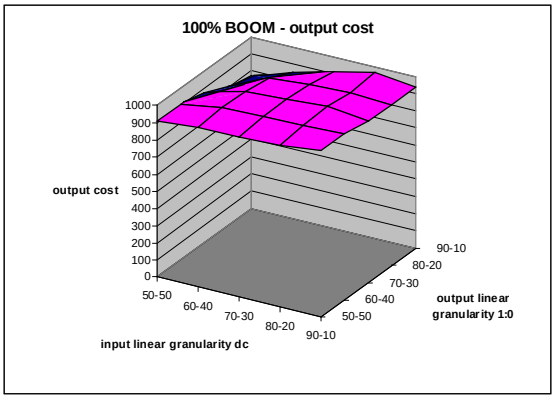
Obrázek 4.6.8 Test č.5 100% FC-Min - terms



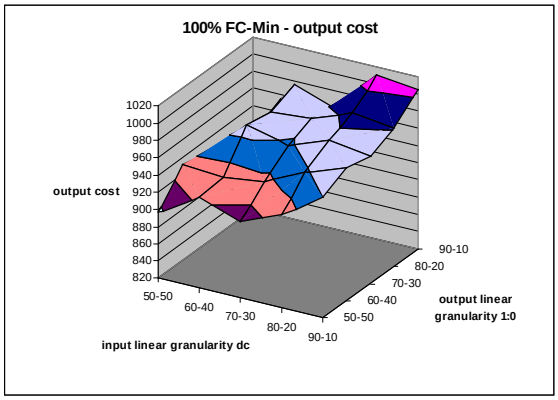
Obrázek 4.6.9 Test č.5 50% BOOM - terms

Zde opět BOOM vykazuje jistou závislost na granularitě vstupních don't care a to že stoupá počet výstupních termů se zvyšující se granularitou a to až skoro na dvojnásobek. FC-Min naopak zde vykazuje opačnou tendenci a to, že se rostoucí granularitou don't care na vstupu klesá počet termů, ale pouze o nějakých 15%. Závislost na granularitě 1:0 na výstupu nebyla nalezena.

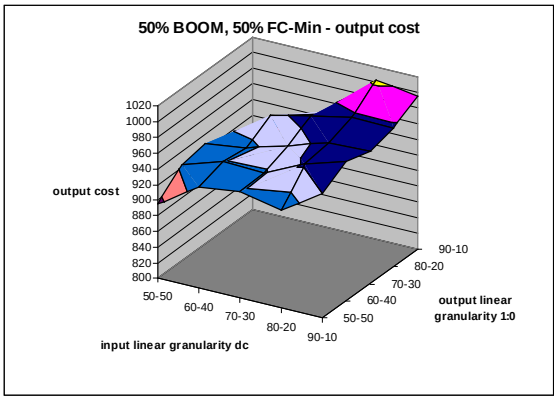
Output cost



Obrázek 4.6.10 Test č.5 100% BOOM - output cost



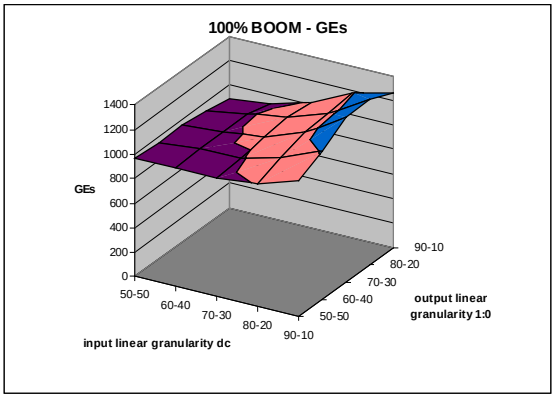
Obrázek 4.6.11 Test č.5 100% FC-Min - output cost



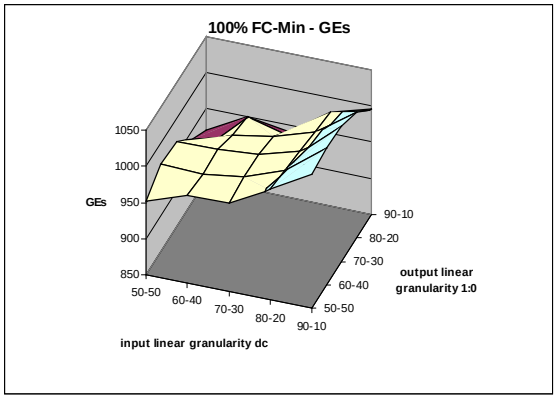
Obrázek 4.6.12 Test č.5 50% BOOM - output cost

Minimalizátor BOOM se v tomto kritériu chová nezávisle na žádné měněné proměnné. FC-Min podobně jako v kríteriu čas vrací dosti nevyrovnané výsledky s velkou rolí náhody. Avšak jistá malá závislost v output cost tu je. Nejhorší výsledky vrací při velké granularitě na vstupních don't cares zároveň s velkou granularitou 1:0 na výstupu, tato odchylka činí přibližně 10%.

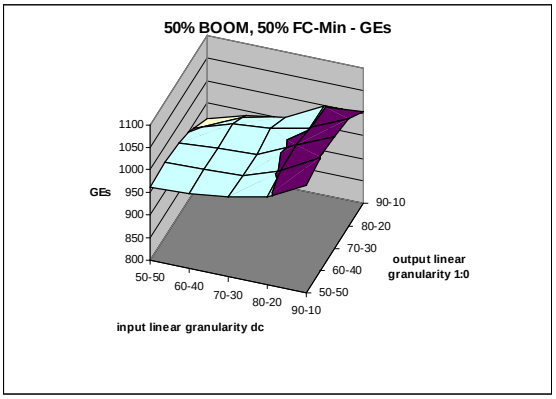
GEs



Obrázek 4.6.13 Test č.5 100% BOOM - GEs



Obrázek 4.6.14 Test č.5 100% FC-Min - GEs



Obrázek 4.6.15 Test č.5 50% BOOM - GEs

U BOOMu se s rostoucí granularitou vstupních don't cares zhoršuje výsledek a to až o 20%. Podobně FC-min také ztrácel s velkou granularitou, ale pouze přibližně 10%. Jinak oba dva minimalizátory vykazovaly podobné výsledky a nelze určit který byl lepší.

4.6.3 Závěr

Tento test ukázal, že granularita nikterak výrazně neovlivňuje chování obou minimalizátorů. Pouze BOOM pracuje pomaleji a s menší jakostí veškerých výstupů při granularitách don't care na vstupu větších než 80-20. Oba dva minimalizátory podávali srovnatelně kvalitní výstup ve všech kategoriích kromě času a počtu termů. V čase byl FC-Min skoro o 2 řády rychlejší a generoval o 20% až 100% méně termů. Opět nastavení poměru spolupráce BOOM a FC-Min vedlo k očekávanému správnému kompromisu mezi oběma variantama.

4.7 Celkový závěr z testů

Myslím že všechny testy dopadly vcelku podle očekávání. Ve většině testovaných případů dopadl lépe minimalizátor BOOM, ovšem s řádově delšími výpočetními časy. Naopak FC-Min je neuvěřitelně rychlý minimalizátor, který se přibližuje výsledku už od první iterace. Ve velké většině testů kombinace obou dvou minimalizátorů věrně průměrovala výsledky každého zvlášť. Domnívám se že při větším počtu iterací by se měla spíše blížit lepšímu výsledku než pouze průměrovat BOOM a FC-Min.

Nejzajímavější bylo testování poměrně specifických vlastností jako je vliv grnularity na výsledek. Občas to přineslo zajímavé náznaky, které by stály za další zkoumání a testování. Jmenovitě například v testu číslo 4. kde se zkoumal vliv různého poměru jedniček a nul jak na vstupu tak na výstupu. Zde docházelo, že BOOM při vetším počtu jedniček než nul na výstupu pracoval lépe než obraceném poměru.

Zklamáním naopak bylo pro mne to, že granularita nepřinesla skoro žádný vliv na testované minimalizátory.

5 Seznam literatury

[1] W.V. Quine: The problem of simplifying truth functions, Amer. Math. Monthly, 59, No.8, 1952, pp. 521-531

[2] E.J. McCluskey: Minimization of Boolean functions, The Bell System Technical Journal, 35, No.5, Nov. 1956, pp. 1417-1444

[3] J. Hlavička and P. Fišer, "BOOM - a Heuristic Boolean Minimizer", Proc. ICCAD-2001, San Jose, Cal. (USA), 4.-8.11.2001, 439-442

[4] P. Fišer, J. Hlavička a H. Kubátová, "FC-Min: A Fast Multi-Output Boolean Minimizer", Proc. Euromicro Symposium on Digital Systems Design (DSD'03), Antalya (TR), 3.-5.9.2003, pp. 451-451

[5] P. Fišer, H. Kubátová, "Two-Level Boolean Minimizer BOOM-II", Proc. 6th Int. Workshop on Boolean Problems (IWSBP'04), Freiberg, Germany, 23.-24.9.2004, pp. 221-228

[6] Coudert, O.: Two-Level Logic Minimization: An Overview, Integration. The VLSI Journal, 17-2, pp. 97-140, Oct. 1994.

[7] V. Jáneš, J. Douša, "Logické systémy", skriptum ČVUT Praha, 2001

[8] http://cs.felk.cvut.cz/~fiserp/projects.html

A. Uživatelská příručka

Syntaxe:

generate -i n -o n -t n [-h n] [-idc n] [-odc n] [-odt n] [-idt n] [-glx n] [-IN] [-NC] [-L] [-timeout n] [file_name]

-i *n*	Nastav počet vstupních proměnných na *n.*
-o *n*	Nastav počet výstupních proměnných na *n.*
-t *n*	Nastav počet generovaných termů na *n*.
-h *n*	Nastav počet bitů použítých v hashovací funkci na *n*. Standartní hodnota je 0. Maximální hodnota je 16.
-idc *n*	Nastav počet don't cares ve vstupní matici na *n* %. Standartní hodnota je 0%.
-odc *n*	Nastav počet don't cares ve výstupní matici na *n* %. Standartní hodnota je 0%.
-idt *n*	Nastav poměr 1:0 ve vstupní matici na *n* %. Standartní hodnota je 50%.
-odc *n*	Nastav poměr 1:0 ve výstupní matici na *n* %. Standartní hodnota je 50%.
-glx *n*	Nastav lineární granularitu, za x lze doplnit idc - nastav druhou mez počtu don't cares ve vstupní matici na *n* %. odc - nastav druhou mez počtu don't cares ve výstupní matici na *n* %. idt - nastav druhou mez poměru 1:0 ve vstupní matici na *n* %. odt - nastav druhou mez poměru 1:0 ve výstupní matici na *n* %. při použití bude první generovaný term mít valstnost uvedenou v x, poslední term bude mít vlastnost uvedenou v glx.
-IN	Vypiš informace o funkci.
-NC	Neprovádí se kontrola konzistence a generuje se PLA typu fd.
-L	Nahraj tabulku uvedenou ve file_name a vypiš o ní info.
-timeout *n*	Nastav timeout na *n* sekund. Standartní hodnota je 1000 s
file_name	výstupní soubor. Pokud není specifikován PLA tabulka je vypsána na standartní výstup.

Tabulka A.1 Popis přepínačů programu generate

Program generuje náhodnou booleovskou funkci ve formátu PLA typu fr nebo typu fd. Standartně je generována tabulka typu fd a tudíž je prováděn test konzistence. Aby nedošlo zařazení některého z termů zaroveň do on-setu a off-setu. Tímto při generování funkcí s velkým procentem vstupních don't care velmi silně stoupá spotřebovaný výpočetní výkon. Proto je v programu nastaven timeout standartně na 1000 vteřin po kterých se program ukončí bez výsledného výstupu.

Příklady

generate -i 100 -o 20 -t 50 -idc 20 out.pla

Generuje PLA tabulku typu fd se 100 vstupními proměnnými, s 20ti výstupními proměnnými. Tabulka bude mít 50 termů a 20% don't cares na vstupu. Tabulka se uloží do souboru out.pla

generate -i 50 -o 20 -t 100 -idt 75 -glidt 25 -IN

Generuje tabulku na standartní výstup s 50ti vstupními proměnnými, se 100 termy a s 20ti výstupními proměnnými. První generovaný term bude mít 75% jedniček na vstupu. Poslední generovaný jich bude mít 25%, zbytek budou nuly. Po ukončeníse na standartní výstup vytisknou statistické informace o vygenerované PLA tabulce.

Poznámka

Při generování lineární granularity don't cares je mnohem výhodnější ji zadávat na vstupní matici sestupně. Např je lepší volit přepínače -idc 80 -glidc 20, nežli opačně. Na výstupní matici je tomu obráceně - je lepší volit menší procento u přepínače -odc.

B. Obsah CD

V kořenovém adresáři se nachází binární spustitelný soubor generate.exe a podadresáře:

src - zde jsou uloženy zdrojové soubory ke generátoru

texty - složka obsahuje tuto práci v elektronické podobě

testy - obsahuje veškeré materiály týkající se provedených testů včetně všech vstupních dat, skriptů na převod formátů, výstupních dat a grafů