Poděkování

Tímto bych chtěl poděkovat vedoucímu své bakalářské práce, Ing. Petru Fišerovi, za pomoc a veškerý čas, který mi věnoval.

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 právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 23.8.2007 ……

Anotace

Cílem této práce je implementovat program pro vizualizaci binárních rozhodovacích diagramů(BDD). Diagram je generován z funkce pomocí balíku CUDD. Funkce jsou zadány pravdivostní tabulkou ve formátu PLA. Práce nejdříve osvětlí základní pojmy týkající se grafů a BDD, poté objasní některé konvence a metody kreslení grafů a nakonec vysvětlí metodiku použitou při tvorbě programu. Na závěr nechybí otestování programu a porovnání s jinými implementacemi.

Abstract

The goal of this bachelor theses is to implement a program for visualization of binary decision diagrams (BDD). The diagram is generated from a function using the CUDD package. The functions are written in truth tables (PLA files). The bachelor theses first explains some theory related to graphs and BDD, then interprets some conventions and methods of graph drawing and finaly explains methodics used in program. At the end I have made some test and compared the result with other implementations.

Obsah

Seznam Obrázků

Seznam tabulek

Úvod

Binární rozhodovací diagramy(BDD) jsou datové struktury vhodné pro reprezentaci, analýzu a testování logických funkcí. Jejich nespornou výhodou je velice úsporné vyjádření, což následně zlepšuje další operace s nimi. Převedením logické funkce na BDD a jeho následnou vizualizací získáváme možnost poměrně jednoduchým způsobem zjistit ohodnocení funkce, viz kapitola 2.2.

Svou bakalářskou práci jsem rozdělil do pěti kapitol. V první kapitole se zaměřuji na nejdůležitější pojmy úzce související s touto prací. Druhá kapitola pojednává o problémech spojených s kreslením grafů; ve třetí popisuji balíček CUDD a práci s ním. V následující kapitole vysvětluji použité algoritmy a podrobně rozebírám datové struktury. V poslední kapitole se pokusím podat a zhodnotit naměřené výsledky. Výsledky své bakalářské práce shrnuji v závěru.V příloze najdete příklady diagramů generovaných výslednou aplikací a pro porovnání diagramy generované pomocí jiných aplikacích zabývajících se vizualizací grafů.

1. Základní pojmy

V této kapitole bych chtěl zmínit některé ze základních pojmů týkajících se následující látky a standardy, které budou dále použity.

1.1. Graf

Graf je obecně struktura, která je tvořena uspořádanou dvojicí množiny vrcholů V a množiny hran E. Řešení mnohých úloh, jako např. nalezení pořadí pro vykonání na sobě závislých úloh nebo nalezení nejkratších silničních spojů mezi městy, se dá převést na grafový problém, jenž má obecné řešení.

Grafy lze rozdělit podle těchto základních hledisek:

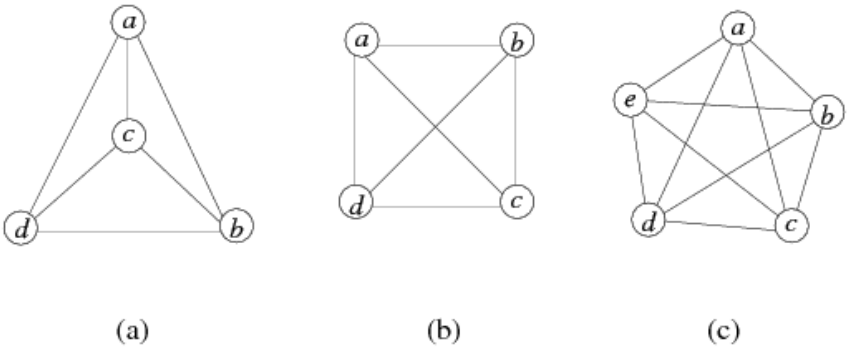
- Podle četnosti hran na jednoduché grafy a multigrafy. V jednoduchém grafu vede mezi jednotlivými vrcholy vždy nejvýše jedna hrana, u multigrafů může být těchto hran více.

- Podle orientace hran na orientované a neorientované. U orientovaných grafů jsou dvojice vrcholů uspořádány, tzn. hrana z E_1 do E_2 je odlišná od hrany z E_2 do E_1. Orientace se často znázorňuje pomocí šipek.

- Podle existence kružnic v grafu na cyklické a acyklické.

- Podle toho, zda lze graf nakreslit do roviny bez křížení hran na rovinné(planární) a nerovinné, viz obr 1.1.

- Podle souvislosti na souvislé a nesouvislé. V souvislém grafu existuje cesta mezi každou dvojicí vrcholů. Silně souvislý graf je navíc takový orientovaný graf, v němž existuje spojení[1] mezi všemi dvojicemi vrcholů.



Obr 1.1 a) planární graf b) neplanárně nakreslený graf z (a) c) neplanární graf

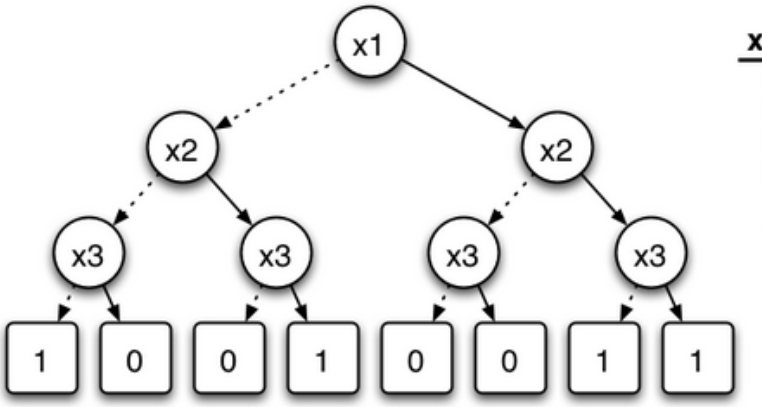
1.2.Binární rozhodovací diagramy (BDD)

BDD je kořenový orientovaný acyklický graf reprezentující logickou funkci. Je to speciální případ rozhodovacího diagramu, jenž se skládá se z jednoho nebo více kořenů, z neterminálních "rozhodovacích" uzlů a jednoho nebo dvou terminálů.

Z takového diagramu lze celkem snadno vyčíst ohodnocení reprezentované funkce. Každý neterminální uzel reprezentuje určitou vstupní proměnnou funkce. Začínáme tvořit cestu grafem od kořene. Pokud nám jde o ohodnocení, v němž je tato proměnná true, pokračujeme hranou HI(then), jinak pokračujeme hranou LOW(else). Tímto způsobem se dostaneme až k terminálu, jehož hodnota je výsledek funkce. Na obrázku 1.2 je příklad jednoho takového BDD.

[1] Spojení - cesta, jež zachovává orientaci hran

Podrobnější informace je možné nalézt na [7].



x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Obr. 1.2 příklad binárního rozhodovacího diagramu

1.3.Formát PLA

PLA je jednoduchý formát pro načítání a ukládání logických funkcí do souboru. Je použit pro načítání BDD. Soubor PLA obsahuje v hlavičce mnoho klíčových slov, zde uvádím pouze ty nejdůležitější, kompletní seznam viz [3].

.i [n] - povinný parametr specifikující počet vstupních proměnných

.o [n] - povinný parametr specifikující počet výstupních proměnných

.ilb [s_1] [s_2] …[s_n] - nutné zadat za obě předešlá klíčová slova(.i , .o), pojmenovává vstupní proměnné

.ob [s_1] [s_2] …[s_n] - nutné zadat za obě předešlá klíčová slova (.i, .o), pojmenovává výstupní proměnné

.e - nepovinné klíčové slovo značící konec popisu PLA souboru

Všechna tato klíčová slova kromě .e jsou uvedena před definicí samotné funkce. Ta je zadána po jednotlivých termech v tomto tvaru:

0-011 001

kde znaky před mezerou určují vstupní proměnné funkce a znaky za mezerou určují výstup termu. Znak pomlčky označuje neurčitý výstup. Příklad PLA souboru je na Obr. 1.1. Jsou na něm popsány dvě funkce:

$$X(A,B,C) = (A \wedge B \wedge C) \text{ a}$$

$$Y(A,B,C) = (\overline{A} \wedge \overline{B} \wedge C) \vee (\overline{A} \wedge B \wedge \overline{C})$$

```
.i 3 .ilb A B C
.o 2 .ob X Y
000 00
001 01
010 01
011 01
10- 00
110 01
111 10

.e
```

Obr. 1.3: příklad PLA souboru

1.4.Postskript

Postskript (PS) je specializovaný programovací jazyk určený pro popis stránek založený na principu abstraktního vícezásobníkového procesoru. Byl vyvinut v roce 1984 firmou Adobe Systemes inc. speciálně pro účely grafických aplikací. Díky poměrně dlouhodobé existenci je dostupný na všech důležitých platformách(PC, Unix, Mac).

Jazyk byl postupem času rozšířen na tři zpětně kompatibilní verze.

Plná definice jazyka PS je značně rozsáhlá, zde zmíním pouze základní příkazy, které vysvětlím na příkladu. Podrobnější informace o PS naleznete na [4].

Souřadnicový systém PS má počátek v levém dolním rohu a základní jednotkou je jeden typografický bod, jehož velikost je rovna 1/72 palce.

Velmi důležitým elementem v PS je tzv. cesta. Jedná se o množinu bodů, úseček a křivek, které jsou navzájem spojeny[1]. Z těchto cest mohou být pak vykresleny obrysy, popř. obrysy vyplněné či kombinace obojího.

Základní příkazy jazyka:

Newpath – dokončí aktuální cestu a vytvoří novou

Moveto – přesune aktivní bod cesty na zadané souřadnice

Lineto - přidá cestě úsečku z aktuálního bodu do cílového

Rlineto - přidá cestě úsečku na zadané relativní souřadnice

Rmoveto - přesune aktivní bod cesty na zadané relativní souřadnice

Stroke - nakreslí aktuální cestu nastaveným stylem

Showpage- odešle aktuální stránku na výstupní zařízení a připraví novou prázdnou stranu

Translate - pracuje s transformační maticí, určuje posunutí následujících souřadnic

Scale - upraví transformační matici, zadává měřítko výkresu

Def - pojmenovává posloupnost příkazů, čímž vlastně vytváří procedury

[1] cesta může obsahovat i mezery

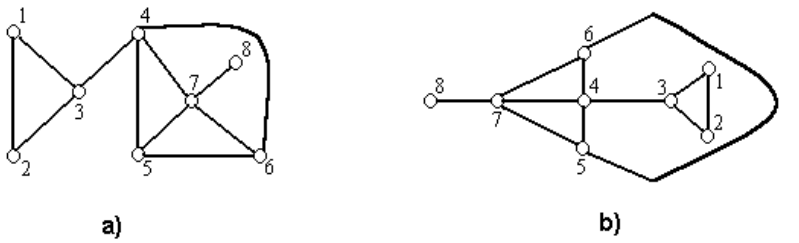
Na Obr. 1.2 je příklad Postskript souboru. Příklad demonstruje vytváření návěští na posloupnost příkazů(vytváření funkcí). Nakreslí čtverec o velikosti 300 bodů. Za povšimnutí stojí hlavně to, že parametry volaných příkazů jsou uvedeny před jmény funkcí. To vyplývá z faktu, že PS je zásobníkově orientovaný jazyk.

```
/box {
400 100 lineto
100 400 lineto
100 100 lineto
400 400 lineto
} def
newpath
100 100 moveto
/box
stroke
showpage
```

Obr. 1.2: příklad Postskript souboru

2. Vizualizace grafů

Problém s vizualizací grafů leží v tom, jak reprezentovat množinu hran a uzlů způsobem, aby nám neuniklo nic z významu, co graf reprezentuje. Je mnoho způsobů, jak nakreslit graf, přičemž každý může být nakreslen s důrazem na jiné spojení a vztahy mezi uzly grafu.



Obr. 2.1 příklad grafu nakresleného různými způsoby

Z obrázku 2.1. a) není tolik patrné, že vrchol 4 má mnoho sousedů a není tak jasně vidět, že vrcholy 3 a 7 odděluje pouze jeden vrchol.

Jak je vidět na Obr. 2.1, matematicky stejný graf zprostředkovává dvě mírně odlišné informace. Úkolem kreslení grafů je tedy maximalizovat míru této informace.

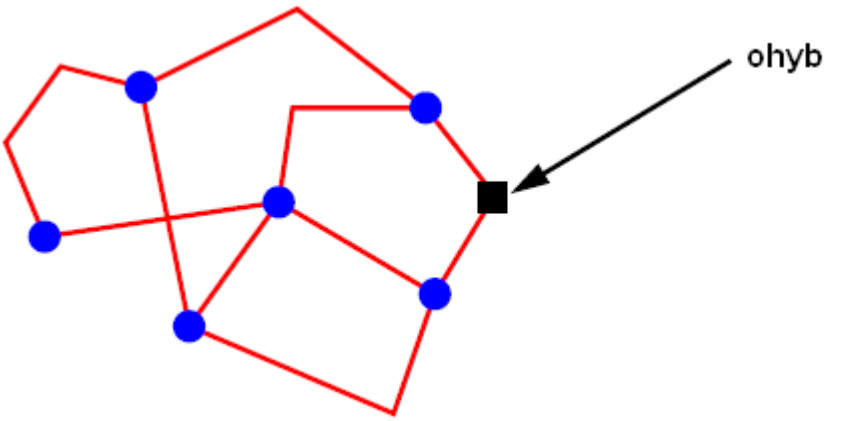
2.1.Konvence Kreslení grafů

Pro jednotlivé okruhy aplikací kreslení grafů se používají různé konvence. V závislosti na účelu objektů jsou uzly typicky reprezentovány body či čtverci a hrany čárami či křivkami. Dále uvádím několik nejběžnějších stylů kreslení grafů:

- Planární kreslení - hrany jsou reprezentovány jednoduchými

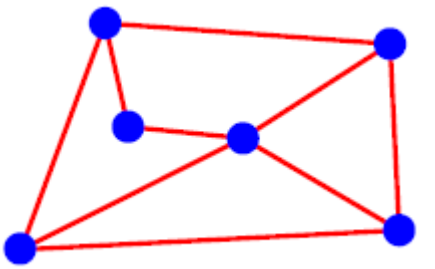
čárami tak, že se žádné dvě hrany nekříží. Ne každý graf jde ovšem nakreslit jako planární viz kap.1.1.

- Kreslení z lomených čar (Polyline Drawing) - hrany jsou reprezentovány sledem po sobě jdoucích úseček. Příklad na Obr. 2.2.



Obr. 2.2 kreslení z lomených čar

- Kreslení z přímých čar (Strightline Drawing) – je kreslení grafů, kde je každá hrana jednoduše reprezentována přímou úsečkou. Příklad na Obr. 2.3.

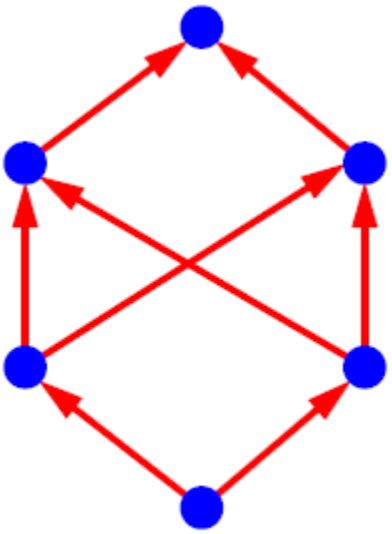


Obr. 2.3 kreslení plánárního grafu z přímých čar

- Ortogonální kreslení – hrany v ortogonálním kreslení jsou vždy pouze vodorovné nebo svislé. Ortogonální grafy se

aplikují např. ve VLSI nebo databázových schématech.

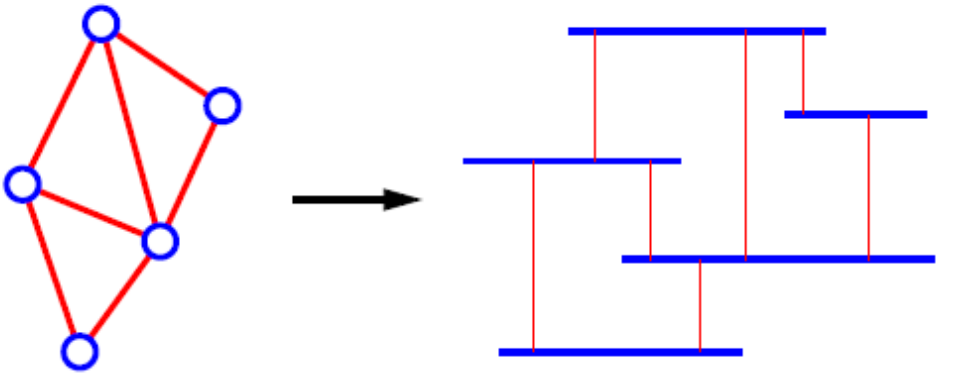
- Vzestupné kreslení – vzestupné kreslení se používá pro kreslení orientovaných grafů. Všechny hrany v tomto grafu musí být směřovány jedním směrem[1].



Obr. 2.4 vzestupné kreslení

- Viditelnostní reprezentace(Visibility representantion)- používá se jako předstupeň pro určité algoritmy. V grafu jsou všechny uzly reprezentovány vodorovnou čarou a hrany svislou čarou. Viz Obr. 2.5.

[1] nemusí být nutně směřovány směrem nahoru



Obr. 2.5 převedení grafu do Viditelnostní reprezentace

2.2. Estetická kritéria

Pro hodnocení, jak je graf čitelný, bylo odvozeno několik kriterií. Mezi ně patří:

- symetričnost – pokud to aktuální aplikace dovoluje, je výhodné zobrazovat symetričnost.

- křížení – minimalizace celkového počtu hran je žádoucí. V ideálním případě lze dostat graf bez křížení hran, ovšem toho nelze dosáhnout u všech grafů.

- délka hran – je podstatné usilovat o co nejmenší délku hran

- plocha – je požadována minimalizace plochy potřebné pro nakreslení grafu. V praktických aplikacích je důležité vytvářet úsporné grafy, neboť se šetří místo na obrazovce či jiném mediu.

- ohyby – je žádoucí minimalizace počtu ohybů

Obecně nelze optimalizovat všechna kritéria najednou. Pokud například vyžadujeme co nejmenší délku hran, může se to projevit na počtu překřížených hran nebo na symetričnosti.

2.3.Přístupy ke Kreslení grafů

Obecně neexistuje jednoduchý způsob, jak nakreslit graf, a ani algoritmus, který by nakreslil „nejlepší“ graf. Nicméně existuje několik postupů pro kreslení grafů, kde jsou popsány základní kroky, jak graf nakreslit. Všechny kroky těchto postupů jsou poté nahrazeny jednotlivými algoritmy, jež se zaměřují na různá estetická kritéria nebo na jiné vhodné rysy.

2.3.1.Topologie – Tvar – Metrika (Thopology – Shape - Matrics)

Výsledné výkresy tohoto přístupu jsou ortogonální grafy, které se, kromě jiného, často používají např. pro elektrotechnická schémata. Výhoda tohoto přístupu je také v minimalizaci překřížených hran, poněvadž první krok je planarizace grafu.

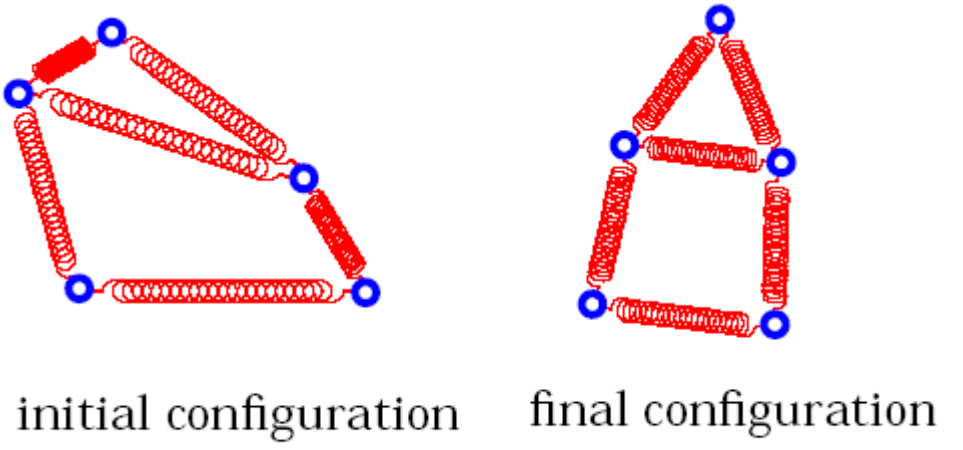
2.3.2.Hierarchický přístup

Tento přístup je výhodný v tom, že zachovává hierarchii acyklických orientovaných grafů. Má využití např. při kreslení diagramů tříd. Tvorba takového grafu se skládá ze tří kroků: V prvním kroku se přiřadí jednotlivé uzly do úrovní tak, že pokud vede hrana z uzlu U do uzlu V, pak V je v nižší vrstvě než U. Navíc pokud je rozdíl mezi vrstvy těchto uzlů větší něž jedna, umístí se mezi ně

tzv. falešný uzel (dummy). V další fázi se redukují počty překřížených hran. V posledním kroku se redukuje počet ohybů vyjímáním falešných uzlů a zkracují se vzdálenosti mezi uzly.

2.3.3.Silově řízený přístup (force based layout)

Silově řízený přístup je nedeterministická metoda používající analogii z fyziky pro kreslení přímočarých grafů. Tento algoritmus funguje tak, že každá hrana je modelována jako pružina s určitou elasticitou. Navíc všechny uzly se odpuzují, jako kdyby byly elektricky nabité. V tomto systému se pak nalezne rovnovážný stav, čímž se dosáhne výsledného grafu. Obr. 2.6 ilustruje daný algoritmus. Výhoda tohoto přístupu je, že neklade žádné nároky na vstupní graf, jako je např. ortogonalita. Další výhodou je jeho přizpůsobivost; lze používat i jiné než elastické a elektrické konstanty. Bohužel má i nevýhody, z nichž nejhorší je poměrně vysoká výpočetní náročnost.



Obr. 2.6 optimalizace diagramu pomocí Silově řízené metody

3. CUDD (Colorado University Decision Diagram)

Existuje mnoho bezplatných balíku pro práci s BDD. Jedním z nich je i CUDD. Podporuje kromě BDD také algebraické rozhodovací diagramy (ADD) a rozhodovací diagramy s potlačenou nulou (ZDD). Jeho předností je poměrně velké množství funkcí a vysoce optimalizovaných algoritmů pro práci s diagramy. Celý balík je volně ke stažení na adrese ftp://vlsi.colorado.edu/pub/cudd-2.4.1.tar.gz. Návod na instalaci a zprovoznění je k dispozici v manuálu [1]. Jak je uvedeno v dokumentaci, balík lze použít několika způsoby:

- Tzv. Černá skřínka: V tomto případě aplikace určená pro práci s rozhodovacími diagramy pouze používá exportované funkce tohoto balíčku popsané v nápovědě. Postačující sada těchto funkcí umožňuje většině programátorům psát tímto stylem své aplikace a nemusí se zaobírat metodikou uspořádávání proměnných.

- Průhledná skřínka: Tento přístup vyžaduje vyšší znalost vnitřních datových struktur CUDDu. Vyplatí se při psaní náročnějších projektů zabývajících se problematikou rozhodovacích diagramů. Vzhledem k efektivitě je nutné přidat do balíku vlastní funkci namísto použití stávajících exportovaných a vnitřních funkcí.

- Rozhraní: Balíček obsahuje také rozhraní pro objektově orientované jazyky jako C++, Perl5 a Java, čímž umožňuje

programátorům nestarat se o správu paměti. Rozhraní C++ je zahrnuto přímo v distribuci. Rozhraní Java a Perl5 je distribuované samostatně.

Základní práci s CUDDem si osvětlíme na příkladu:

```
void main(){
    DdManager *manager;
        DdNode *f, *var, *tmp;
        int i;
        ...
        f = Cudd_ReadOne(manager);
        Cudd_Ref(f);
        for (i = 3; i >= 0; i--) {
            var = Cudd_bddIthVar(manager,i);
            tmp = Cudd_bddAnd(manager,Cudd_Not(var),f);
            Cudd_Ref(tmp);
            Cudd_RecursiveDeref(manager,f);
            f = tmp;
        }
```

Před začátkem práce s knihovnou je nutné prvně inicializovat strukturu. DdManager voláním funkce Cudd_Init(manager). Samotný BDD diagram se tvoří od spodu, tzn. prvně pomocí volání funkce Cudd_ReadOne(manager).

4. Implementace

V této kapitole budou popsány detaily samotné implementace programu pro vizualizaci BDD nazvaného VizBDD. Nutno podotknout, že zde uvedené výpisy zdrojových souborů jsou z důvodů lepší čitelnosti často zkráceny. Jejich kompletní verzi najdete na CD přiloženém k této publikaci.

4.1.Použité nástroje

Jako jeden z bodů zadání práce je začlenění funkcí pro zobrazování BDD do balíčku CUDD. Jelikož je tento balíček v rámci co největší optimalizace napsán v jazyku C, zvolil jsem tentýž jazyk. Nicméně výsledná aplikaci je napsána pro 32bitovou platformu Windows, v níž se neobejdeme bez jazyku C++. Výsledný kód je tedy z části napsán v C a z části v C++.

Aplikace využívá knihovny balíku CUDD. Tento balík je primárně určen pro platformu Linux. V projektu jsou ovšem použity knihovny upravené O. Kološem [2] pro požití na platformě Windows, který také přidal funkce pro načítání ze souboru typu Pla.

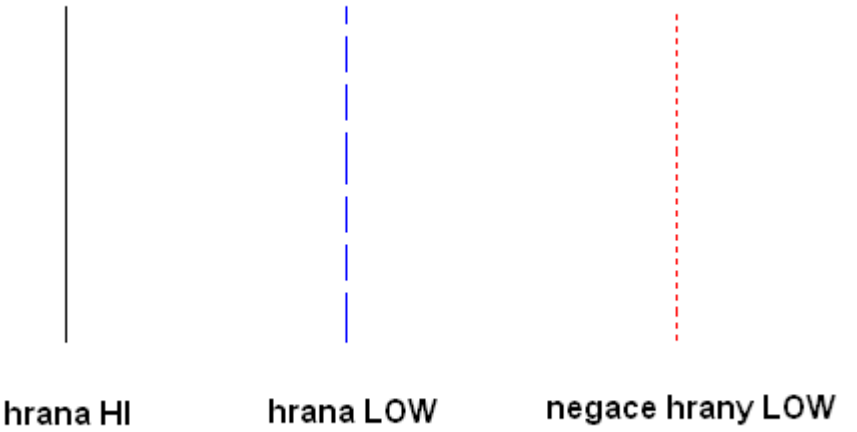
Celý projekt je vytvořen ve vývojovém prostředí Visual Studio 2005. Nicméně funkce pro načtení grafu, optimalizaci grafu a uložení do formátu PostScript jsou psány čistě v C a nejsou v nich použity žádné systémové závislosti. Přenos těchto funkcí na jiné platformy by tedy neměl být žádný problém.

4.2.Program VizBDD

Na následujících řádcích bude popsáno základní ovládání výsledné aplikace - programu VizBDD.

Vstupem programu je soubor ve formátu PLA. Před otevřením souboru je dobré zvážit, kterou metodu optimalizace grafu vybrat a počet iterací této optimalizační metody. Principy metod budou vysvětleny níže (4.4), nicméně jejich volba výrazně ovlivňuje dobu nutnou pro načtení grafu. Defaultně je zvolena metoda Barycenter. Metodu lze vybrat z hlavního menu programu, v sekci Control.

Graf je generován s koncovým terminálem “1“. Černá plná čára značí hranu hi (then), modrá čárkovaná čára značí hranu low (else) a červená tečkovaná čára značí negovanou hranu low. Na obrázku 4. 1. je ukázka tohoto značení.



.

Obr. 4.1 ukázka značení hran

Poslední dvě možnosti programu jsou klasické možnosti skrolování grafu v okně a zoom. Při zoomování se nemění velikost písma. S jeho pomocí lze tedy změnit výstupní soubor, neboť mění poměr délek hran s velikostí jednotlivých uzlů.

4.3.Hodnotící funkce

Součástí zadání je také vymyslet nějakou funkci, která by vyjadřovala, jak je daný graf čitelný a jestli je jeden graf „lepší“ než druhý. K tomuto účelu byla odvozena jistá estetická kritéria viz kapitola 2.2. Kvůli vyšší přehlednosti grafu jsem zvolil hierarchický přístup, čímž všechny uzly mají pevně danou souřadnici y podle proměnné, jež reprezentují. Vzhledem k hierarchickému se některá kritéria těžko ovlivní, a proto byla vypuštěna. Jedná se například o celkovou plochu nutnou k vykreslení. Tato plocha je tedy v tomto případě dána počtem literálů(souřadnice Y) a nejvyšším počtem výskytu jednoho literálu v termech(souřadnice X).

Dále je důležité, že se grafy nedají optimalizovat pro všechna estetická kritéria najednou. Pokud například budeme vyžadovat co nejmenší délku hran, projeví se to na symetričnosti, nebo pokud chceme minimalizovat počty křížení hran (např. zavedením ohybů), projeví se to na délce hran.

Další narok na tuto hodnotící funkci je, aby byla výpočetně co nejméně složitá. Je to z toho důvodu, že se používá v heuristickém algoritmu (algoritmus GREEDY kapitola 5.4.2), a tak je opakovaně volána.

Nakonec jsem tedy zvolil tři kritéria, podle nichž graf hodnotit:

- Celková délka všech hran D
- Počet překřížených hran C
- Počet hran, jež překreslují uzel P

Výslednou funkci jsem stanovil na:

$$\frac{\left(Y \cdot C + D\right) \cdot \left(P + 1\right)}{U}$$

Kde U je počet uzlů v grafu a Y je šířka grafu (též počet literalů-2).

4.4.Použité algoritmy

Pro generování grafu jsem implementoval tři různé algoritmy.

4.4.1.Metoda Simply

Metoda Simply načítá graf z BDD. Jedná se vlastně jen o vhodné umísťování uzlů do grafu bez další optimalizace. Algoritmus postupně prochází celý BDD od všech kořenů do hloubky. Kontroluje se, zda-li již uzel nebyl přidán a pokud ne, tak se nový uzel umístí vždy na zvyšující se souřadnici X v pořadí inorder. Poté se vždy vpravo a vlevo do pater, mezi nimiž vede hrana, umisťují tzv. falešné uzly (dummy).

Takto generovaný graf sice není úplně optimální a pro větší počet vstupních logických funkcí se roztahuje do šířky, nicméně se díky falešným uzlům vyhneme problému kreslení hran přes uzly. Výhodou zmiňované metody je především její rychlost. Zbylé metody tuto funkci využívají k načtení grafu z BDD.

4.4.2.Metoda Greed

Jedná se o tzv. hladový algoritmus využívající hodnotící funkci, jež je součástí zadání, viz 4.3. Před spuštěním algoritmu se graf načte pomocí výše zmíněné metody Simply. Graf se postupně prochází po řádcích a uzly se posouvají vpravo. Pokud se výsledek hodnotící funkce zmenšil, změna se zachová. To se opakuje dokud se během iterace uloží alespoň jedna změna, nebo dokud není překročen maximální počet iterací. Poté se procházejí falešné uzly a “zkouší” se vyjmout. Opět se při tom kontroluje zlepšení hodnotící funkce a při zlepšení se změna zachová.

4.4.3.Metoda Barycenter

Tato metoda minimalizuje jak počet křížení hran, tak délku samotných hran. Před spuštěním metody je nutné načíst graf, … Metoda se skládá ze tří fází. V první fázi metoda spočítá pro určitou vrstvu horní těžiště jednotlivých uzlů a přehází uzly tak, aby byly seřazeny podle těchto těžišť. Zkontroluje se počet překřížených hran, a pokud je nižší než před změnou, změna se zachová. To se opakuje postupně pro jednotlivé řádky směrem dolů. Poté provádí to samé s tím, že se řadí podle spodních těžišť a postupuje po řádcích směrem nahoru. Tyto dva postupy se opakují, dokud se během průchodu aspoň jednou uzle prohodí, nebo dokud není překročen maximální počet iterací.

V druhé fázi se opět prochází graf po řádcích dolu a nahoru, a pokud se najdou uzly se stejným těžištěm, prohodí se a spustí se fáze jedna. I tato fáze se opakuje, dokud není překročen maximální

počet iterací.

Nakonec se redukují délky hran. To se provádí tak, že se graf opět prochází směrem dolu a nahoru a prvně se umisťují falešné uzly pod resp. nad místa svých předchůdců resp. potomků. Poté se opět projde graf od shora dolů a nahoru a uzle se umístí na nejbližší volné místo od těžiště směrem doleva.

4.5.Funkce importované do CUDDu

Zde budou popsány hlavní funkce, jež jsou vytvořeny jako jakési rozšíření balíku CUDD. Pro použití těchto funkcí je nutné přikompilovat hlavičkový cuddGraph.h a knihovnu cuddGraph.a. Jak je v CUDDu zvykem, pojmenoval jsem začátky importovaných funkcí “Cudd_”. Vytvořené funkce mají jako první parametr ukazatel na strukturu GRAPH, v níž je uložená vnitřní reprezentace grafu, viz. 4.6.2. Deklarace těchto funkcí je uvedena na Obr. 4.2.

```
extern void Cudd_GraphSavePs(GRAPH *g, char *path);
extern bool Cudd_GraphMake(GRAPH *g, DdManager *manager,
                DdNode **f, int out, int in, char **oNames, char **iNames);
extern double Cudd_GraphEvaluate(GRAPH *g, bool ignored);
extern void Cudd_GraphDelete(GRAPH *g);
extern void Cudd_GraphOptimalize(GRAPH *g, int method, int iter);
```

Obr. 4.2 deklarace importovaných funkcí

Popis funkcí:

- Funkce Cudd_GraphMake vytvoří a vygeneruje graf. Funkci je nutné zavolat před jakýmikoliv dalšími operacemi s grafem. Vysledný graf je vygenerován algoritmem, jenž je popsán jako metoda Simply. Jako první parametr vyžaduje ukazatel na struktru GRAPH. Tato struktura nesmí být alokována, funkce si ji alokuje sama. Druhý parametr je ukazatel na strukturu DdManager, která obsahuje BDD, jež se má vykrestlit. Třetí parametr je pole ukazatelů na kořeny BDD. Čtvrtý a páty parametr udává počet vstupních resp. výstupních proměnných

funkcí výstupů. Parametr oNames je pole ukazatelů na názvy výstupních proměnných, parametr iName je pole ukazatelů názvů vstupních proměnných.

- Funkce Cudd_GraphSavePs uloží diagram do souboru typu PostScript. Druhý parametr je cesta cílového soboru.

- Funkce Cudd_GraphEvaluate ohodnotí graf pomocí výše zmíněné hodnotící funkce viz. kapitola 4.3.

- Funkce Cudd_GraphOptimalize optimalizuje graf. Druhý parametr udává funkci, jednu z dvou metod, jež se má pro optimalizaci použít. Možné hodnoty tohoto parametru jsou MD_GREED (viz. kapitola 4.4.2) a MD_BARYCENTERS (viz. kapitola 4.4.3). Třetí parametr udává maximální počet iterací, které se v těchto algoritmech využívají. Provedení optimalizace může v závitslosti na velikosti grafu a na počtu iterací trvat až několik minut.

- Funkce GraphDelete smaže graf a uvolní alokovanou paměť.

4.6.Popis datových struktur

V programu jsou kromě jiných použity tři hlavní datové struktury: NODE, GRAPHINFO, GRAPH a jedna třída WinGraph. Soubor vizBdd.cpp obsahuje funkce pro ovládání okna. Používá rozhraní windows api, podrobnosti naleznete v [11].

```
typedef struct Node{
    char *name;
    Node *pThen;
    Node *pElse;
    POINT loc;
    int type;
    bool comp;
}NODE, *PNODE;
```

Obr. 4.3 deklarace datové struktury node

4.6.1.Datová struktura NODE

Deklarace této struktury je uvedena na Obr. 4.3.

- V této datové struktuře jsou uloženy všechny informace nutné pro zobrazení jednoho uzlu. Položka name obsahuje ukazatel na jméno uzlu.

- Položky pThen a pElse jsou ukazately na následníky uzlu. Pokud se ale nejedná o klasický typ uzlu, ale o falešný uzel, jejich význam je trochu odlišný. V tomto případě, kdy je potomek jenom jeden, položka pElse stále ukazuje na potomka, pThen je ale využita tak, že ukazuje na první uzel v řadě, který není falešný.

- Položka loc obsahuje aktuální souřadnice, kde je uzel v grafu umístěn.

- Položka type vyjadřuje, o jaký uzel se jedná. Nabýva hodnot: 0 pro normální typ uzlu, 1 pokud je uzel kořen, 2 pokud se jedná o falešný uzel, jenž reprezentuje hranu HI, a 3 pokud jde o falešný uzel hrany LOW.

- Položka comp je nastavena na true, pokud je hrana LOW negovaná.

4.6.2.Datová struktura GRAPH

Hlavní struktura reprezentující celý graf se jmenuje GRAPH. Obsahuje úplnou reprezentaci grafu a několik přidružených proměnných vyjadřujících určité vlastnosti vykreslovaného grafu. Deklarace je na Obr. 4.4 . Význam jednotlivých proměnných je následující:

```
typedef struct {
    NODE ***graph;
    POINT size;
    POINT shift;
    NODE *one;
    INSNODE *inserted;
    unsigned int grid;
    unsigned int border;
}GRAPH, *PGRAPH;
```

Obr. 4.4 deklarace datové struktury graph

- Položka graph je nejdůležitější proměnná v této struktuře. Je to ukazatel na dynamicky alokované dvojrozměrné pole ukazatelů na NODE. Pomocí tohoto pole jsou všechny uzly umístěny v grafu. Dochází sice k redundantně uložené informaci, neboť souřadnice jednotlivých uzlů jsou dány jak indexy v tomto poli, tak položkou loc, nicméně tento systém značně urychluje přístup k informaci.

- V proměnné size je uložena velikost grafu.

- Položka shift udává posunutí začátku vykreslovaného grafu.

- Položka one je ukazatel na jediný terminální uzel grafu.

- Položka grid udává velikost buňky mřížky, na níž se umisťují uzly. Tato velikost je v pixelech.

- Položka inserted je ukazatel na spojový seznam. Tento seznam obsahuje ukazatele na všechny uzle v grafu a urychluje tak v situacích, kdy není vyžadován hierarchický přístup procházení grafem. Kromě toho je také využíván pro vyhledávání již vložených uzlů z BDD:

4.6.3.Třída WinGraph

Třída WinGraph je použita jako prostředník mezi oknem aplikace VizBDD a knihovnou graph. Obsahuje funkce pro ovládání tohoto okna. Její deklarace je na Obr. 4.4. Význam důležitějších funkcí je následující:

- WinGraph je konstruktor třídy. Jako parametr má ukazatel na okno aplikace, s nímž celá třída pracuje.

- Draw funkce vykreslí graf do okna.

- LoadPLA funkce inicializuje BDD, volá funkce pro natažení BDD ze souboru, jehož cesta je jako první argument. Vytvoří graf a zavolá optimalizaci grafu. Jelikož je optimalizace značně časově náročná, vytváří pro ni nové vlákno.

- Funkce SavePs volá funkci pro ukládání do souboru a jako parametr má cestu k tomuto souboru.

- Funkce Zoom mění měřítko, v němž se zobrazuje graf. Změna tohoto měřítka je argumentem této funkce.

- Proměnná graph je jediná instance struktury GRAPH

- hWnd je ukazatel na okno, jenž WinGraph ovládá

```
class WinGraph
{
public:
    WinGraph(HWND window);
    ~WinGraph(void);
    void Draw();
    bool LoadPLA(TCHAR *path, int method, int iter);
    void Scroll(int nBar, int to, unsigned int step=1);
    void Resize(int wParam);
    void SavePs(TCHAR *path);
    void Zoom(int scale);
    bool IsWorking(){return working;};
private:
    GRAPH graph;
    int iHorzPos, iVertPos;
    bool working;
    HPEN penThen;           //pero jimz se vykresluji Then hrany
    HPEN penElse;           //pero jimz se vykresluji Else hrany
    HPEN penComp;           //pero jimz se vykresluji negovane hrany
    SCROLLINFO si;
    RECT drawRect;
    HANDLE workThread;
    HWND hWnd;
};
```

Obr. 4.5 deklarace třídy WinGraph

5. Testy

V této kapitole zkusím podat a zhodnotit výsledky, jež jsem naměřil při použití výsledné aplikace VizBDD. Všechny testy jsou prováděny na počítači Intel Celeron D 2.66 Mhz s 768 MB operační paměti a operačním systémem Windows XP. Jako testovací data jsem použil několik souborů z MCNC benchmarku.

V tabulce 5.1 jsou uvedeny časy metod Greed a Barycenter, a hodnocení grafů pro jednotlivé metody. Jelikož metoda Simply neprovádí žádnou optimalizaci, časy generování grafu se pohybovaly pod 1s a proto nejsou uvedeny.

Pomalost metody Greed je způsobena dvěma faktory. Za prvé je to celkem pomalé zpracovávání hodnotící funkce[1] která je periodicky volána. Za druhé je to nutnost velkého počtu opakování, aby bylo dosáhnuto optimalizovaného grafu.

Metoda Barycenter dosahuje ve slušném čase poměrně dobré výsledky. Bohužel má jednu nevýhodu a to, že každá iterace nepřináší vždy lepší výsledek. Tento postřeh je doložen tabulkou 5.2.

Název testovaného souboru	Počet vstupních / výstupních proměnných	Počet uzlů v grafu	Metoda				
			Simply	Greed		Barycenter	
			ohodn.	čas[min:s]	ohodn.	čas[min:s]	ohodn.
alu.pla	12/8	21	56	3	29	1	32
sex.pla	9/14	51	128	21	72	2	64
root.pla	8/5	58	102	4	66	1	51
t1.pla	21/23	134	235	8:47	133	10	126
al2.pla	16/47	139	199	16:44	151	12	163
in7.pla	26/10	235	195	22:10	160	34	144

Tabulka 5.1 časová složitost optimalizačních algoritmů

V posledním testu je porovnána doba vygenerování a optimalizace grafu programem Graphviz [8]. V příloze pak najdete

[1] asymptotická složitost hodnotící funkce $O(H^2)$ kde H je počet hran v grafu

ukázky grafů, vygenerované jednotlivými metodami a programem Graphviz.

Počet iterací	Barycenter	ohodnocení
1	2	150
2	6	156
3	13	157
5	36	156
10	2:25	151
20	13:11	152

Tabulka 5.2 časová složitost a hodnocení grafu pomocí metody Barycenter v závislosti na počtu iterací (vstupní soubor al2.pla)

Název souboru	Graphviz	VizBDD
mark1.pla	8	1:57
in7.pla	4	35
t2.pla	2	9
al2.pla	3	12

Tabulka 5.3 porovnání s programem Grapwiz

Závěr

Podařilo se mi splnit všechny body zadání, tzn. BDD jsem vykreslil, vymyslel hodnotící funkci vyjadřující čitelnost grafu a začlenil možnost vizualizace do balíčku CUDD. Podařilo se mi úplně vyhnout křížení uzel - hrana, čímž jsem docílil lepší přehlednosti výsledného diagramu.

Když výsledný program porovnám s jinými podobnými aplikacemi, na grafech o menších nebo středních velikostech se výsledky moc neliší. Grafy s větším počtem uzlů(přibližně 200 a více) se bohužel poněkud dlouho načítají a jsou nečitelné. Nicméně s takovými grafy ostatní aplikace neporadí o mnoho lépe.

Dalších možností, jimiž by se dal výsledný program vylepšit, je například zavedení kreslení pomocí (bezierových) křivek, přidáním dalších výstupních formátů nebo podporou tisku.

Literatura

[1] Colorado University Decision Diagram Package manual
http://vlsi.colorado.edu/~fabio/CUDD/

[2] Kolář, J.: Teoretická informatika. (skripta), Praha, Česká informatická společnost, 2004, ISBN 80-900853-8-5

[3] http://service.felk.cvut.cz/vlsi/prj/BOOM/pla_c.html

[4] PostScript language reference manual / Adobe Systems Incorporated.3rdd.
http://www.adobe.com/products/postscript/pdfs/PLRM.pdf

[5] Encyklopedie Wikipedia http://www.wikipedia.org

[6] Graph Drawing: Algorithms for the Visualization of Graphs by Ioannis G. Tollis

[7] O. Kološ: „port programového balíku CUDD pod platformu Windows“ Bakalářská práce na FEL ČVUT 2006

[8] Graphviz - Graph Visualization Software
http://www.graphviz.org

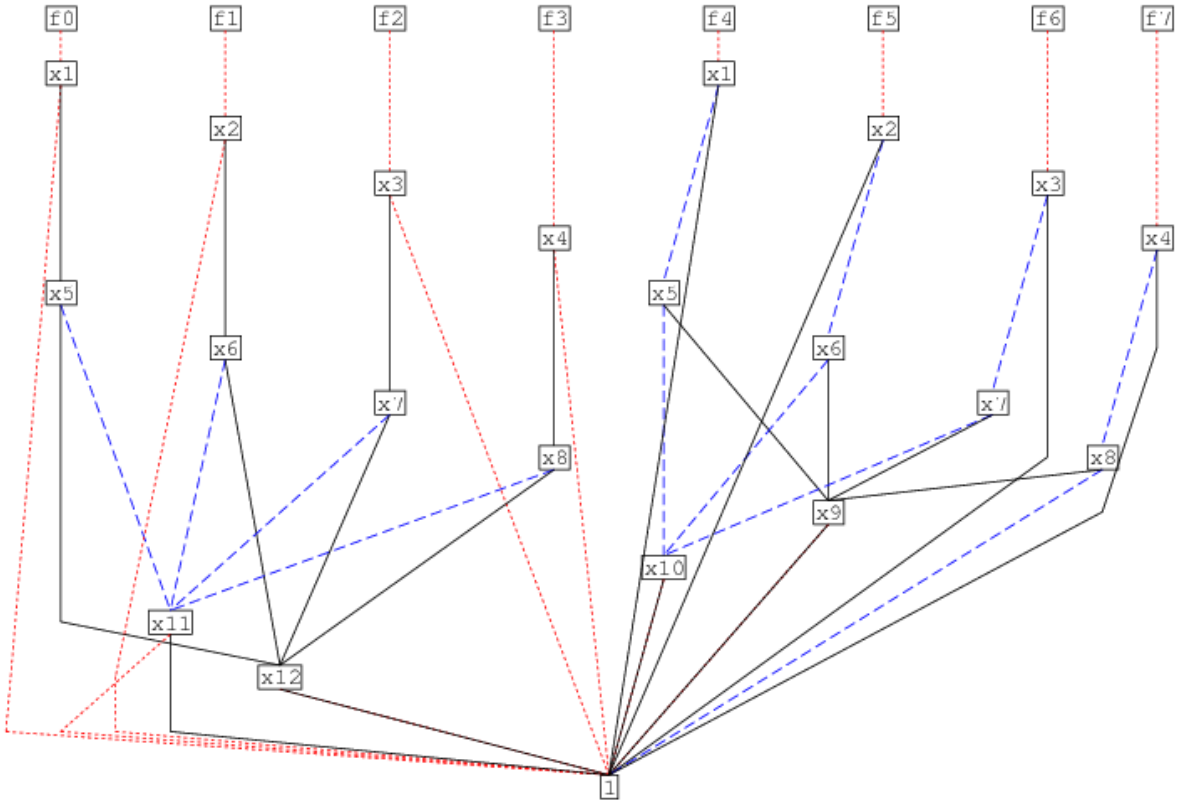
[9] Md. Adul Hassane Samee: „Upward Planar Drawings of Directed Acyclic Graph“

[10] Methods for Visual Understanding of Hierarchical System Structures Kozo Sugiyama, Shojiro Tagawa, Mitsuhiko Toda Presented by Ivan Bowman March 12, 1998 http://plg.uwaterloo.ca/~itbowman/CS746G/ASug.html

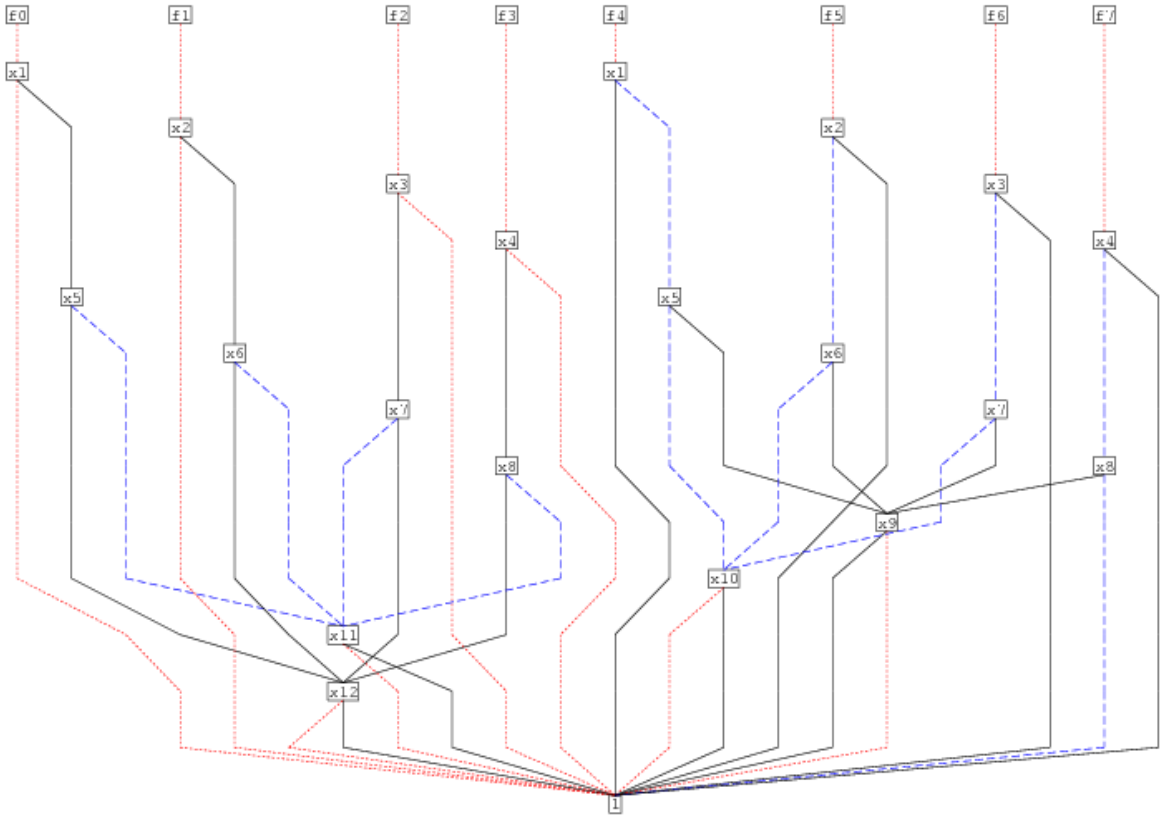
[11] Charles Petzold: Programování ve Windows. Vydání první, Compter Press 1999.ISBN 80-7006-206-8

Příloha A

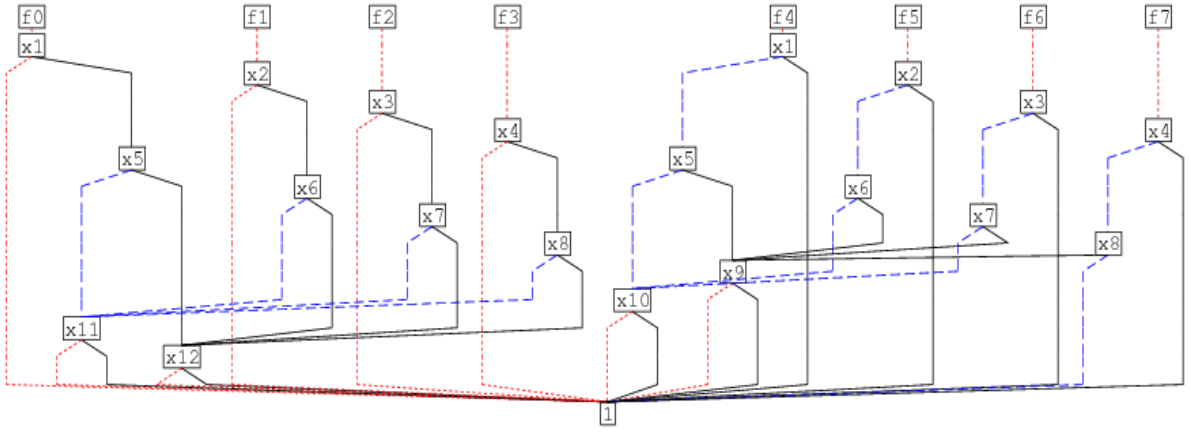
Ukázky vygenerovaných grafů



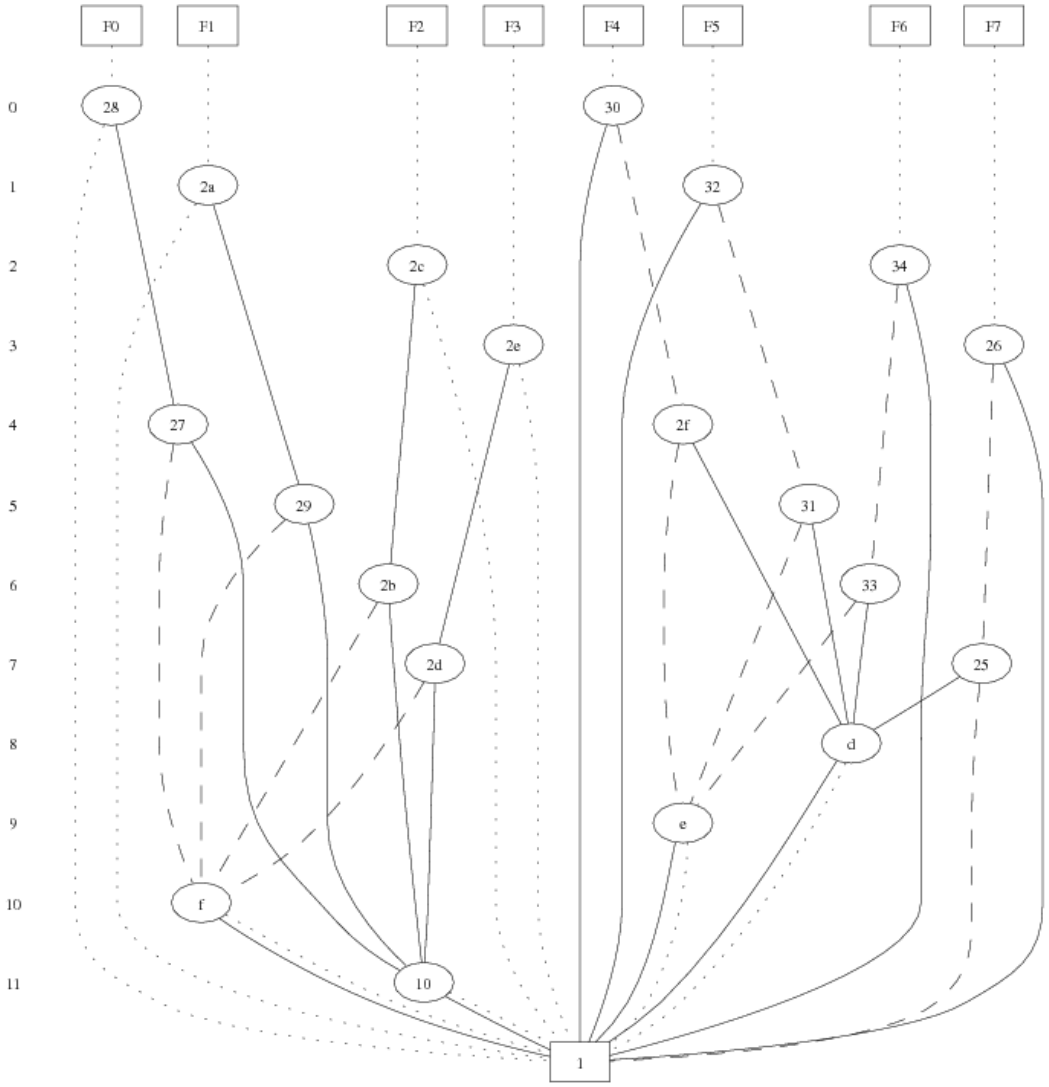
Obr. A.1 graf Alu1.pla vygenerovaný pomocí metody greed



Obr. A.2 graf Alu1.pla vygenerovaný pomocí metody greed



Obr. A.3 graf Alu1.pla vygenerovaný pomocí metody simply



Obr. A.4 graf Alu1.pla vygenerovaný pomocí aplikace Graphviz

Příloha B

Obsah CD

- **Kořenový adresář –** v tomto adresáři naleznete kromě všech dalších adresářů spustitelný soubor VizBDD.exe

- **Dokumentace** – v tomto adresáři naleznete elektronickou verzi této práce

- **Zdrojový_kód** – adresář se zdrojovými kódy. V podadresáři Projekt pro MSVC2005 naleznete projekt pro Microsoft Visual Studio 2005

- **Testovací_soubory** – obsahuje testovací soubory