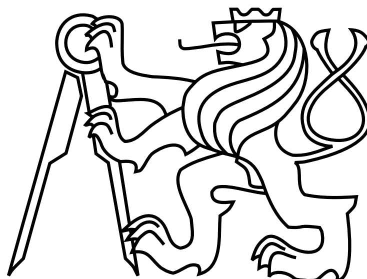
České vysoké učení technické v Praze
Fakulta elektrotechnická



Diplomová práce

Generátor náhodných logických obvodů

Bc. Tomáš Měchura

Vedoucí práce: Ing. Petr Fišer

Studijní program: Elektrotechnika a informatika dobíhající magisterský

Obor: Výpočetní technika

květen 2008

Poděkování

Rád bych poděkoval ing. Petru Fišerovi za ochotu a vůli při pomoci s vypracováním této práce. Dále mým přátelům a kamarádům za jejich pochopení mé neúčasti na společných akcích. A v neposlední řadě lidem, kteří utvářeli atmosféru, při které tato práce vznikala.

Prohlášení

Prohlašuji, že jsem svou diplomovou 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 23.5.2008 ....

Abstract

This diploma thesis deals with implementation of three different generators of randomized logic circuits. The first one generates Boolean functions in the form of PLA table. The second one deals with generating of final state machine in KISS format considering the state machine determinism. The last program is focused on combinational logic circuit made of simple gates. The circuit is then optimised with genetics algorithms. In the end of thesis the third program is tested how input parameters affects the circuit's testability and resynthesis.

Abstrakt

Tato diplomová práce pojednává o tvorbě tří různých generátoru náhodných logických obvodů. První generuje popis obvodu formou Booleovských funkcí v tzv. PLA formátu. Druhý program řeší generování náhodného konečného automatu ve formátu KISS s ohledem na jeho determinismus. Třetí program se zabývá tvorbou kombinačního obvodu z jednoduchých hradel a jeho následnou optimalizaci pomocí genetických algoritmů. V závěru práce je třetí program testován na vliv nastavitelných parametrů na jeho minimalizaci a testovatelnost.

Obsah

Seznam obrázků xv

Seznam tabulek xvii

Slovník pojmů xix

1 Úvod 1

1.1 Požadavky a cíle . . . 1

2 Analýza a návrh řešení 3

2.1 Formáty výstupních souborů . . . 3

2.1.1 Formát PLA . . . 3

2.1.2 Formát KISS . . . 4

2.1.3 Formát BLIF . . . 5

2.1.4 Formát BENCH . . . 5

2.1.5 Formát SLIF . . . 6

2.2 Použité technologie . . . 6

2.3 genpla — booleovská funkce . . . 7

2.4 genstate — konečný automat . . . 8

2.4.1 Definice . . . 8

2.4.2 Grafové algoritmy . . . 8

2.4.3 Generování PLA pro přechody . . . 11

2.5 gennetwork — síť logických hradel . . . 11

2.5.1 Triviální obvod . . . 11

2.5.2 Optimalizační algoritmy . . . 13

2.5.3 Genetické algoritmy . . . 14

2.5.4 Testovatelnost . . . 18

3 Implementace 23

3.1 genpla . . . 23

3.2 genstate . . . 23

3.2.1 Použité datové struktury . . . 23

3.2.2 Průběh algoritmu . . . 24

4 Implementace gennetwork 25

4.1 Datové struktury . . . 26

4.1.1 Graf zapojení obvodu . . . 26

4.1.2 Entity . . . 27

4.2 Průběh algoritmu 29
4.3 Popis některých důležitých metod 30
4.3.1 Základní operace se spoji a hradly 30
4.3.2 Metoda na tvorbu nového hradla 30
4.4 Generování triviálního obvodu 31
4.5 Genetický algoritmus 32
4.5.1 Mutace obvodu 34
4.5.2 Fitness funkce 36
4.5.3 Ostatní parametry 37
4.6 Testovatelnost 38

5 Testování **41**
5.1 Nastavení testů 41
5.2 Test č.1 42
5.2.1 Nastavení 42
5.2.2 Naměřené výsledky 42
5.2.3 Vyhodnocení 43
5.3 Test č.2 43
5.3.1 Nastavení a metodika testu 44
5.3.2 Naměřené výsledky 44
5.3.3 Vyhodnocení 45
5.4 Test č.3 47
5.4.1 Nastavení 47
5.4.2 Naměřené výsledky 48
5.4.3 Vyhodnocení 51
5.5 Test č.4 51
5.5.1 Generování obvodů 52
5.5.2 Metodika testu 52
5.5.3 Naměřené výsledky 52
5.5.4 Vyhodnocení 54

6 Závěr **55**
6.1 Náměty na vylepšení 55

7 Literatura **57**

A Uživatelská příručka **59**
A.1 genpla 59
A.2 genstate 61

A.3 gennetwork 63

B Obsah přiloženého CD 67

Seznam obrázků

2.1 Graf stavového automatu 9

2.2 Průběh genetického algoritmu 15

2.3 Různé druhy křížení 17

2.4 SCOAP: Výpočet řiditelnosti 19

2.5 SCOAP: Výpočet pozorovatelnosti 20

5.1 Test č.1 - Průběh fitness funkce pro různá nastavení velikosti populace . 44

5.2 Test č.2 - Četnost "influence" - testovatelnost 0 46

5.3 Test č.2 - Četnost "influence" - testovatelnost 0.1 46

5.4 Test č.2 - Četnost "influence" - testovatelnost 0.2 46

5.5 Test č.2 - Závislost "influence" na "ratiu" obvodu 47

5.6 Test č.3 - Četnost "influence" - 10 výstupů - 30 hradel 50

5.7 Test č.3 - Četnost "influence" - 10 výstupů - 150 hradel 50

5.8 Test č.3 - Četnost "influence" - 10 výstupů - 600 hradel 50

Seznam tabulek

5.1 Test č.1 - nastavení parametrů 43

5.2 Test č.1 - výsledky 43

5.3 Test č.2 - Parametry generátoru 45

5.4 Test č.2 - Výsledky měření 45

5.5 Test č.3 - Parametry generátoru 48

5.6 Test č.3 - Naměřené výsledky - 1 výstup 48

5.7 Test č.3 - Citlivost výstupu v jednotlivých vzorcích - 1 výstup 49

5.8 Test č.3 - Naměřené výsledky - 10 výstupů 51

5.9 Test č.4 - Parametry obvodů ze souboru ISCAS 53

5.10 Test č.4 - Naměřené výsledky 54

A.1 Popis přepínačů programu genpla 59

A.2 Popis přepínačů programu genstate 61

A.3 Popis přepínačů programu gennetwork - část 1. 63

A.4 Popis přepínačů programu gennetwork - část 2. 64

Slovník pojmů

ATPG Nástroj na automatické generování testovacích vektorů a následné otestování obvodu na detekovatelné a nedetekovatelné poruchy.

Literál proměnná nebo její negace.

Term vyjádření součtu nebo součinu literálů.

P-Term součinový term neboli součin literálů.

S-Term součtový term neboli součet literálů.

genpla 1. generátor zaměřený na výstup ve formátu PLA.

genstate 2. generátor zaměřený na tvorbu stavových automatů.

gennetwork 3. generátor zaměřený na tvorbu logických sítí tvořených z kombinačních hradel.

fan-in počet vstupů hradla

fan-out počet výstupů hradla

1 Úvod

Cílem této práce bylo navázání na mojí bakalářskou práci - Parametrizovaný generátor náhodných booleovských funkcí [10] a upravit stávající generátor opravením několika chyb a přidáním několika funkcí.

Dalším úkolem bylo rozšířit jeho funkčnost na tvorbu konečných automatů. Zde se zaměřit především na možnost tvorby deterministických automatů. Dále pak implementovat metody, aby generovaný automat byl spojitý, aby se v něm nenacházely stavy ze kterých nelze nikam přejít a aby se v automatu neobjevili nedostupné stavy.

V neposlední řadě bylo za úkol vytvořit generátor víceúrovňového logického obvodů formou zapojení jednoduchých kombinačních hradel. Tento úkol byl od počátku zjednodušen volbou, že se v obvodě nebudou nacházet zpětné vazby. Program měl brát v úvahu složitější parametry jako celková testovatelnost obvodu, maximální větvení všech interních spojů a počet úrovní obvodu. Bylo nutné vygenerovat nejdříve náhodný triviální obvod s pevnými parametry a ten posléze zkusit měnit některou optimalizační metodou tak, aby co nejvíce odpovídal zadaným parametrům.

Nakonec bylo důležité tento generátor obvodů otestovat na vliv jeho některých vstupních parametrů a na reálnou testovatelnost obvodu. Především na počet nedetekovatelných poruch, zjištěných nástrojem ATPG. Dále pak na obtížnost resyntézy tvořených obvodů pomocí nástrojů na minimalizaci. K tomuto účelu mi byl poskytnut nástroj SIS [5].

1.1 Požadavky a cíle

Zde uvádím finální požadavky na všechny tři části generátoru, které se v průběhu práce po konzultacích drobně měnily s nově objevenými komplikacemi.

U testování byl kladen důraz na ověření funkčnosti a použitelnosti vytvořeného generátoru, na vyvození správných zavěrů a navrhnutí dalšího pokračování.

1.část — generátor obvodu ve formátu pravdivostní tabulky (genpla)

- opravit chyby v již hotovém programu, způsobující jeho pády
- doplnit o možnost kontroly duplicitních termů
- zefektivnit práci programu

2.část — generátor obvodu ve formátu konečného automatu (genstate)

- základní parametry obvodu — počet vstupů, počet výstupů, počet stavů, počet přechodů.
- možnost generování ekvivalentních stavů
- možnost generování deterministického i úplného automatu
- možnost různé metody na ošetření dostupnosti/spojitosti stavů automatu
- ukládaní automatu ve formátu KISS

3.část — generátor obvodu sítí hradel (gennetwork)

- zadané neměnné parametry obvodu — počet vstupů, počet výstupů a počet hradel
- v obvodech nebudou zpětné vazby
- obvod bude tvořen základními hradly (AND, NAND, OR, NOR, XOR, XNOR, NOT)
- hradla budou vícevstupová a jejich maximalní stupeň půjde nastavit
- libovolný spoj v obvodě nepřesáhne zadaný stupeň větvení
- obvod se bude generovat s ohledem na testovatelnost a na počet úrovní obvodu
- výstup obvodu ve formátech BENCH a BLIF

Testování

- otestovat vygenerované sítě hradel s ohledem na reálnou testovatelnost vypočítanou nástrojem ATPG.
- ověřit funkci obvodů jejich resyntézou pomocí nástroje na minimalizaci SIS.
- zjistit z testů případné závislosti na vstupních parametrech generovaných obvodů.

2 Analýza a návrh řešení

Jelikož je úkolem vytvořit generátor, který generuje 3 odlišné formy logického obvodu, rozhodl jsem se vytvořit 3 oddělené programy. Ty se budou moci specializovat jen na daný problém a tím pádem ho efektivněji řešit. První část problému již řeší program **genpla**, generátor řešící druhou část jsem vytvářel pod názvem **genstate** a poslední generátor se jmenuje **gennetwork**. Pro spravný návrh všech funkcí programů a hlavně jejich vnitřní implentaci je potřeba důkladně projít všechny požadavky a řádně zvážit všechny možnosti řešení. Proto zde nejprve popíši formáty výstupních souborů, které z nemalé části ovlivnily tuto volbu.

2.1 Formáty výstupních souborů

Všechny zde popisované a používané formáty jsou textové. To znamená, že jsou při prohlížení souboru pro člověka čitelné bez jakýchkoliv úprav.

2.1.1 Formát PLA

Formát je používán ve výstupu genpla. Popisoval jsem ho již v Bakalářské práci [10]. Jelikož je důležitý i pro pochopení dalšího formátu používaného v genstate, popíši ho i zde. Kompletní popis lze stáhnout ve zdorjovém kódu ESPRESSa na [3].

PLA tabulka se skládá z klíčových slov počínající tečkou a jedné 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 poté 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

Příklad PLA — 4 vstupy, 2 výstupy, 4 termy:

```
.i 4
.o 2
.p 4
```

```
.type fr
-011 01
1--0 10
1111 00
100- 10
.e
```

2.1.2 Formát KISS

Formát KISS je rozšíření PLA formátu o sekvenční chování obvodu. Je definován v rámci formátu BLIF pro sekvenční obvody. Jeho úplnou specifikaci nalezneme zde [2]. Jde vlastně o PLA tabulku, kde každý term znamená přechod z jednoho stavu do druhého.

Na počátku souboru je několik klíčových slov uvozených tečkou. Dále se nachází postupně na každém řádku jeden term pro jeden přechod v automatu. Řádek obsahuje:

- vektor ohodnocení vstupních proměnných — význam symbolů jako v PLA
- název stavu ze kterého přecházíme
- název stavu do kterého přecházíme
- vektor ohodnocení výstupních proměnných při uskutečněném přechodu — význam symbolů jako v PLA

Ukázka jednoduchého úplného deterministického automatu ve formátu KISS — 1 vstup, 2 výstupy, 3 stavy, 6 termů

```
.i 1
.o 1
.p 6
.s 3
.r START
0   START  state1 1
1   START  state2 0
1  state1   START 0
0  state1  state1 0
0  state2   START 0
1  state2  state1 1
.e
```

2.1.3 Formát BLIF

Tetno formát již popisuje hierarchii kombinačních bloků a jejich vzájemné propojení v logickém obvodě. Každý blok má definovány názvy vstupů a výstupů a logická funkce bloku je definována jako jeho booleovská funkce. K popisu funkce se používá část již vysvětleného formátu PLA. Úplnou specifikaci nalezneme v [2]. Zde vysvětlím jen „pro nás“ důležité věci.

Jako v předchozích případech začíná formát několika klíčovými slovy uvedenými tečkou. Zde jsou uvedeny globální parametry pro celý obvod, včetně názvů všech vstupů a výstupů. Následují postupně vypsané všechny logické bloky v obvodu. Každý blok je uvozen klíčovým slovem ".names", následován na stejném řádku názvy postupně všech spojů, které jsou pro tento blok logickými vstupy a vzápětí i výstupy. Na dalších řádcích je uvedena alfanumerická matice vyjadřující booleovskou funkci daného bloku, jak je uvedeno již ve formátu PLA. Formát je ukončen klíčovým slovem ".end"

Ukázka jednoduchého logického obvodu ze sítí hradel. Obvod má 3 vstupy, 1 výstup, 2 dvouvstupové hradla (NOR+NAND)

```
.model main
.inputs i1 i2 i3
.outputs o1
.names i1 i2 q1
00 1
.names i3 q1 o1
0- 1
-0 1
.end
```

2.1.4 Formát BENCH

Podobně jako BLIF, tento formát popisuje zapojení jednotlivých logických bloků v rámci celé sítě. Ovšem zde se za bloky berou pouze jednoduchá hradla a to AND, NAND, OR, NOR, XOR, XNOR a BUF. Formát je proto velmi jednoduchý a skládá se pouze z názvů vstupů a výstupů obvodu a seznamu hradel s jejich zapojením.

Každý vstup je uveden na nové řádce, kde je jeho název uzavřen v kulatých závorkách za klíčovým slovem "INPUT". Podobně je každý výstup uveden v závorkách za slovem "OUTPUT". Každé hradlo je zapsáno na samostatném řádku kdy jeho výstupu přiřazujeme výslednou funkční hodnotu jeho vstupů. Jako název funkce je uveden název hradla. Nejlépe to bude vidět na následjícím příkladu.

Ukázka jednoduchého logického obvodu ze sítí hradel. Obvod má 3 vstupy, 1 výstup, 2 2-vstupové hradla (NOR+NAND) — stejný příklad jako u formátu BLIF

```
INPUT(i1)
INPUT(i2)
INPUT(i3)
OUTPUT(o1)
q1 = NOR(i1, i2)
o1 = NAND(i3, q1)
```

2.1.5 Formát SLIF

Do programu jsem se rozhodl zavést ještě tento formát, který je velmi podobný formátu BLIF, avšak místo PLA matice k popisu booleovských funkcí daného bloku, používá jejich logický zápis. Konec řádku se zde ukončuje středníkem. Negace se v zápisu logické funkce vyjadřuje jako apostrof za negovaným výrazem.

Ukázka jednoduchého logického obvodu ze sítí hradel. Obvod má 3 vstupy, 1 výstup, 2 2-vstupové hradla (OR+NAND) — stejný příklad jako u formátu BLIF

```
.model main ;
.inputs i1 i2 i3 ;
.outputs o1 ;
q1 = ( i1 + i2 )' ;
o1 = ( i1 i2 )' ;
.end
```

2.2 Použité technologie

Konzolovou aplikaci, jako je navrhovaný generátor bez jakéhokoliv grafického výstupu, lze vytvářet pomocí mnoha procedurálních i objektových programovacích jazyků. Poněvadž byl k vytvoření původního generátoru použit jazyk C++ s prostředím Borland C++ Bulider 6.0, takže výběr jiného jazyka (bez použití stávajícího kódu) se stal téměř nemožným. Zvolil jsem proto neměnit vývojový jazyk a psal jsem v něm oba generátory. Jelikož mi vývojové prostředí Borland není tak blízké, rozhodl jsem se pro vývojové prostředí Microsoft Visual studio 2005. Volba mezi objektovým nebo procedurálním programováním pro mě nebyla moc složitá, poněvadž přínos objektového programování na tak specifický úkol by nebyl nikterak výrazný v porovnání se zvětšením zdrojových kódů při

zapouzdřování objektů. Navíc je mé osobě bližší procedurální přístup. Tudíž z vlastností C++ a objektů je použito jen minimum. V projektech genpla a genstate byli použity některé knihovny a zdrojové kódy z minimalizátoru BOOM se svolením ing. Fišera. Projekt gennetwork byl vytvořen bez přídavných nestandartních knihoven, poněvadž všechny jeho datové struktury a algoritmy jsou implementovány přímo na míru.

Všechny tři generátory jsou tudíž vytvářeny programovacím jazykem C++ v prostředí Microsoft Visual studio 2005.

Jako další pomocné technologie jsem použil programy Rational od firmy IBM. Šlo především o program Purify, který za běhu kontroluje přístupy do paměti a hledá zápisy mimo alokovanou paměť a její správnou dealokaci. Používal jsem ho na předejití závažných těžko objevitelných chyb a především na kontrolu genpla.

2.3 genpla — booleovská funkce

U 1. genrátoru byly požadavky jasné a srozumitelné. Nejprve bylo nutné přenést hotový program do nově zvoleného prostředí a pomocí zvolené technologie, na analyzování paměti IBM Rational Purify, najít a odstranit nečekané pády programu.

Dalším úkolem k vyřešení bylo přidání kontroly duplicitních termů. Zde se nevyskytoval žádný zádrhel a proto bylo zvoleno řešení, které při zapnutí přepínače bude volat další kontrolní metodu, která otestuje přidávaný term vůči již hotovým na duplicitu. Pokud nastane duplicita, tak se term zahodí a zkusí se vygenerovat nový náhodný term.

V poslední řadě bylo za úkol optimalizovat kód. Zde jsem po důkladném prozkoumaní zjistil, že všechny kontroly, zda vygenerovaný term zařadit do výsledku, probíhají postupně. Tudíž se všechny již zařazené termy procházejí několikrát, podle toho které kontroly jsou zapnuté či nikoliv. Řešením tedy bylo veškeré kotroly zahrnout do jednoho průchodu již vytvořenými termy. Poněvadž se kontroly používají při každém vygenerovaném termu a jsou časově úměrné termům již vygenerovaným, tak se zdá, že by tato úprava mohla výrazně zvýšit efektivitu programu.

2.4 genstate — konečný automat

2.4.1 Definice

Podle analýzy výstupního formátu jsem zjistil, že budeme generovat Mealyho synchronní konečný automat. Tzn. že výstup automatu je závislý pouze na vstupních proměnných a stavu ve kterém se automat nachází. Opakem je Mooreův automat kdy výstup automatu je závislý čistě jen na předchozím stavu automatu.

Ekvivalentní stavy

Dále si vysvětlíme co je to ekvivalentní stav. Podle definice jsou dva stavy Q1 a Q2 automatu A ekvivalentní, pokud všechny možné posloupnosti různých kombinací vstupních proměnných, aplikované na automat A v počátečním stavu Q1, produkují na výstupu tytéž odezvy, jako kdyby byly aplikovány na automat A v počátečním stavu Q2. Skupiny ekvivalentních stavů tvoří tzv. třídy ekvivalence. Je zřejmé, že pokud třída ekvivalence obsahuje n stavů, tak je $n - 1$ stavů redundantních a tudíž je můžeme vyřadit. Všechny přechody do vyřazených stavů přepíšeme na přechod do stavu, který nám zbyl jako jediný v třídě ekvivalence. Je jasné, že když tento postup otočíme tak budeme schopni vygenerovat libovolný počet tříd s libovolným počtem redundantních stavů. Tímto byl vyřešen problém s generováním ekvivalentních stavů.

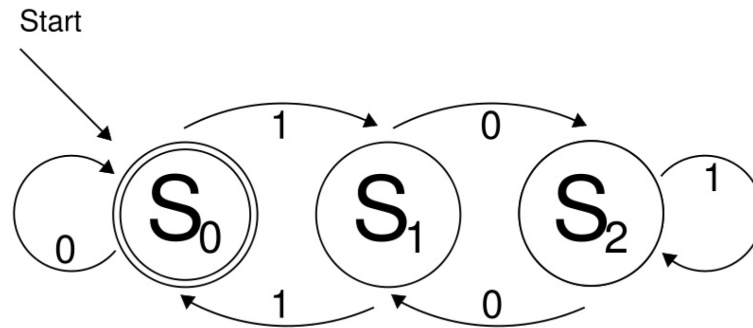
Determinismus a úplnost

Úplný automat je takový, který má v každém svém stavu definován přechod pro všechny kombinace nastavení vstupních proměnných.
Pod pojmem deterministický automat se myslí úplný automat, který nemá v rámci jednoho stavu 2 přechody, které odpovídají stejnému nastavení vstupních proměnných a zároveň mají nastaven odlišný následující stav.
Avšak v rámci této diplomové práce jsem od deterministického automatu nevyžadoval jeho úplnost. Vždy když budu mít na mysli úplný deterministický automat, tak to řádně uvedu.

2.4.2 Grafové algoritmy

Předem uvedu, že stavový automat se dá představit jako orieontovaný graf, kde jedntolivé stavy automatu jsou uzly grafy a přechody mezi nimi jsou reprezentovány jako

orientované hrany grafu.
Při této představě můžeme požadovanou spojitost a jiné parametry automatu úspěšně řešit pomocí znamých grafových algoritmů. Graf úplného deterministického automatu zaobrazuje pro názornost obrázek 2.1



Obrázek 2.1: Graf stavového automatu

Spojitost

Neorientovaný graf je spojitý, pokud se lze z libovolného uzlu grafu pomocí posloupnosti přechodů po jeho hranách, dostat do všech ostatních uzlů grafu.

U orientovaného grafu se analogicky definuje silná spojitost a ta je definována. Orientovaný graf je silně spojitý, pokud se lze mezi všemi uzly grafu pomocí posloupnosti přechodů po jeho orientovaných hranách, dostat do všech ostatních uzlů grafu a zpět.

Spojitost stavového automatu uvedená v požadavcích byla myšlena jako dosažitelnost libovolného stavu ze stavu počátečního. Tudíž nám postačí neorientovaný graf stavů a zajistit jeho spojitost. Jelikož v generátoru musí jít nastavit i počet přechodů, je potřeba zajistit spojitost s co nejmenším možným počtem přechodů. Proto jsem došel k závěru, že bych mohl využít algoritmů pro tvorbu kostry grafu. Tím získám spojitý automat s pouze $n-1$ počtem přechodů, kde n je počet jeho stavů.

Triviální algoritmus na spojení všech prvků náhodnými spoji mi přišel velmi jednoduchý a málo náhodný. Algoritmus pracuje asi takto: Nejprve se utvoří prázdná množina uzlů Q a množina všech uzlů V. Do množiny Q se budou zařazovat již spojené uzly, v množině V jsou doposud nespojené uzly. Algoritmus vždy vybere náhodný prvek z množiny V a vytvoří mezi ním a náhodným prvkem z množiny Q spoj. Poté přeřadí uzel z množiny V do množiny Q. Pokud je množina Q prázdná algoritmus končí.

Vymýšlel jsem proto i jiná řešení založená na předgenerování spojitého grafu trivialním automatem s nadbytkem hran, jejich ohodnocením a poté puštěním algoritmu na tvorbu

minimální kostry grafu.

Minimální kostra grafu

Pro minimální kostru grafu jsou nejznámější 3 lišící se algortimy.

Kruskalův algoritmus

Jde o trivialní algoritmus, budem postupně tvořit množinu hran H s pomocí vzestupně seřazené množiny všech hran E podle ohodnocení.
Postup je v následujících krocích:

- vyjme se z množiny E nejméně ohodnocená hrana k a vloží se do množiny H
- pokud množina H obsahuje kružnici tak se hrana k vyřadí z množiny H

Tento proces se opakuje dokud v množině H není $N-1$ hran, kde N je počet uzlů grafu. Množina H spolu se všemi uzly grafu tvoří minimální kostru.

Jarníkův alglritmus

Zde se vytvoří N množin pojmenovaných $Q1..QN$, kde každá obsahuje právě jeden uzel grafu. Potřebujeme ještě seřazenou množinu hran E podle ohodnocení a jednu prázdnou množinu M. Algoritmus probíhá takto.

- vyjmeme z množiny E první hranu k. Hrana k spojuje uzly U a V
- pokud U a V leží v jiné množině, tak množiny spojíme a hranu k zařadíme do množiny M

Algoritmus pokračuje dokud nám nezbude pouze jedna neprázdná množina Qn. Kostra grafu je určena množinou hran M se všemi uzly původního grafu.

Borůvkův algoritmus

Je velice podobný Jarníkovu algoritmu, ovšem v každé iteraci hledá nejméně ohodnocené hrany, které vedou z každé neprázdné množiny Qi do jiné neprázdné množiny Qj. Poté všechny množiny, mezi kterými byla nalezena hrana, spojí dohromady a všechny tyto hrany přidá do výsledku. Algoritmus pokračuje dokud nezůstane pouze jediná neprázdná množina Qn.

Volba algoritmu

Všechny 3 algoritmy, podávají stejný výsledek a používají seznam řazených hran. Rozdíl je akorát v tom, jakým způsobem hledají hrany na přidání a jak kontrolují, aby se nepřidala přebytečná hrana. Nejvhodnější pro počítačovou implementaci je Jarníkův algoritmus, poněvadž spojování množin je jednodušší, než hledat kružnici z hran. O proti Borůvkovu algoritmu nemusí složitě kontrolovat více hran zároveň. Proto budu implementovat Jarníkův algoritmus.

Koncovost a Dostupnost

Problém, aby do každého stavu a z každého stavu vedl alespoň jeden přechod v celém automatu, je celkem triviální. Nejlepším řešením je projít všechny stavy a doplnit náhodně chybějící přechody.

2.4.3 Generování PLA pro přechody

Ve výstupním formátu jsou přechody automatu definovány rozšířenou PLA tabulkou, proto bude vhodné použít již hotovou 1.část generátoru na generování tabulky, kterou pak doplníme o přechody. Budeme muset generovat N tabulek, kde N je počet generovaných stavů. V případě ekvivalentních stavů, budou tabulky (až na ono doplnění o přechody) identické.

2.5 gennetwork — síť logických hradel

Ze zadaných požadavků je jasné, že nějaká přímo řízená syntéza obvodu, která by generovala obvod se všemi požadovanými parametry, je nemožná. Proto je jasné, že použijeme některé optimalizační metody na dosažení kýžených parametrů obvodu. Při všech optimalizačních metodách je však potřeba vycházet z nějakého již hotového řešení a toto řešení různými metodami modifikovat a zušlechťovat jeho parametry. Proto bylo důležité analyzovat i tvorbu triviálního řešení.

2.5.1 Triviální obvod

Nejprve bylo důležité rozhodnout, které parametry budeme optimalizačními algortimy přizpůsobovat a které budou v průběhu neměnné. Volba těchto pevných parametrů úzce souvisí s volbou optimalizačního algoritmu a obráceně. Generátor by měl umět pracovat s těmito parametry:

- počet vstupů
- počet výstupů
- počet hradel
- typy generovaných hradel
- maximální fan-in jednotlivých hradel
- maximální větvení spojů
- pravděpodobnsot výskytu jednotlivých hradel
- počet úrovní hradel
- celková řiditelnost
- celková pozorovatelnost
- celková testovatelnost

Parametry jsou seřazeny podle jednoduchosti jejich kotroly při syntéze triviálního obvodu. Počet vstupů lze velmi jednoduše kontrolovat a dodržet v průběhu syntézy, nežli globální parametry jako je testovatelnost, které nemohou být známé dokud není celý obvod zapojen.

Od začátku jsem se snažil generovat triviální obvod podle co nejvíce kritérií, abych optimalizačnímu algortimu co nejvíce snížil dimenzi prohledávaného prostoru. Zvolil jsem proto všechny parametry v seznamu, až po maximální větvení spojů-včetně, za pevné. A všechny triviální obvody je museli splňovat. Zároveň byl vybrán a upraven genetický algortimus takový, který by tyto parametry ve svém průběhu již neměnil.

Tím ovšem vznikl problém, že ne všechny kombinace parametrů vytvoří jakkoliv zapojitelný obvod. Uvedu extrémní případ: pro kombinaci 10-ti vstupů, 1 výstupu a 3 dvouvstupových hradel nelze vytovřit obvod. U tohoto případu je to očividné. Ovšem u případů, kde se bude zadávat maximální větvení spojů na nízké hodnoty, již to tak očividné není. V tomto případě nastane při větším počtu hradel problém, že je nebudeme mít kam zapojit. Těchto případů je ovšem tak mizivé množství, že výhody které přinese méně proměnlivých parametrů při optimalizaci, tuto nevýhodu vynahradí. Správné nastavení těchto parametrů bylo přenecháno na rozumu uživatele. Generátor v případě nelogicky zadaných parametrů může uváznout v nekonečné nesplnitelné smyčce, nebo se

může ukončit bez výsledky a v ojedinělých případech může vyvolat vyjímku a ukončit se. Toto záleží v jaké části se objeví při generování problém.

Triviální obvod se také generuje s ohledem na pravděpodobnost výskytu jednotlivých hradel, avšak už je striktně nekontroluje. Možnost kontroly je ponechána optimalizaci.

Při nezadání jiných parametrů na vstupu generátoru, než výše zmíněných pevných paramterů, tak se optimalizační část vůbec nepouští. Použije se přímo vygenerovaný triviální obvod.

2.5.2 Optimalizační algoritmy

Všechny druhy optimalizace jsou založeny na hledání globálního maxima nějaké ohodnocovací funkce, která je závislá na všech optimalizačních kritériích. Počet takto zvolených kritérií nazýváme dimenze stavového prostoru optimalizace. Všechny platné kombinace oněch kritérií vytvářejí celý stavový prostor, ve kterém se hledá nejlepší řešení daného problému. Je vhodné se snažit udržet dimenzi stavového prostoru co nejmenší, poněvadž prostor roste s jeho dimenzí zpravidla exponencionálně. Protože neznáme a ani nejsme schopní dopředu určit které stavy jsou řešením, můžeme vyloučit algoritmy, které systematicky procházejí stavový prostor. Budeme se věnovat jen algoritmům inspirovanými přírodními procesy. Algorimty se vzájemně liší především způsobem procházení tohoto prostoru tzv. pohybem v prostoru. Tyto pohyby se uskutečňují především aplikací nějakých jednoduchých operací na již vytvořená řešení.

Při výběru jsem zvažoval tyto algoritmy :

- Simulované žíhání
- Genetické algoritmy
- Mravenčí kolonie

Při výběru bylo nejdůležitější jestli vůbec dokážeme splnit nároky na procházení stavovým prostorem kladené jednotlivými algoritmy. Simulované žíhání, stejně jako Mravenčí kolonie, klade nároky na tvorbu řešení v pevně zadaném okolí problému, což nemůžeme při tvorbě obvodů nikterak kloudně splnit. Zůstal nám tedy pouze genetický algoritmus, který ovšem klade nároky na schopnost jednoznačně převést — zakódovat řešení problému na jistou posloupnost znaků — tzv chromosom.

2.5.3 Genetické algoritmy

Genetický algoritmus je pokročilá metoda optimalizace založená na principu genetiky, především na přírodním výběru. Jejím principem je udržovat si konstantně velkou sadu řešení — tzv populaci a opakovaným aplikovaním různých metod docílit její evoluci. Rozlišení kvality jedince v rámci populace se provádí fitness funkcí, která převede všechny otpimalizační kritéria na jednorozměrnou proměnnou, která určuje míru splnění těchto kritérií. Nejdůležitější metodou k evoluci populace je selekce silných jedinců. Jejich vzájemným křížením vniká nový jedinec, který má velkou pravděpodobnost, že bude alespoň stejně kvalitní jako jeho rodiče. Dalším důležitým prvkem je přežití nejsilnějších jedinců do další generace. Důležitým aspektem je fakt, že pro správný chod algoritmu, musíme mít hotovou jednu populaci jedinců, kteří řeší problém alespoň triviálním způsobem. Celý algoritmus probíhá asi takto:

1. vygenerování triviální populace
2. zakódování jedinců do podoby chromozomu
3. selekce silných jedinců
4. křížení vybraných jedinců
5. mutace vytvořených jedinců
6. nahrazení původní populace nově vytvořenými jedinci
7. pokud není splněno ukončující kritérium opakuj od bodu 2.

Opakující se fáze je znázorněna na obrázku 2.2. Počáteční populace je vygenerována s pomocí triviálních obvodů popsaných výše.

Kódování jedinců

Problém genetických algoritmů je v tom, že musíme mít popsáno řešení daného problému tzv. chromozómem. Nejlépe se v počítačích prezentuje chromozóm, jako binární pole stejné délky pro všechna řešení. Různá řešení se liší v jednotlivých bitech tohoto chromozómu — genech. V praxi ale nemusí být zakódování tak úplně přesné, pokud k tomu upravíme i zbylé části algoritmu. Za geny tak můžeme volit libovolné prvky, které dohromady reflektují určité řešení problému a mají v rámci různých řešení stejný význam.

Obrázek 2.2: Průběh genetického algoritmu

Při konkrétní aplikaci na problém zapojení obvodů, bylo na snadě volit za geny hradla v obvodu a jako pořadí genu v chromozómu zvolit úroveň odpovídajícího hradla. Tím nám vzniknul chromozóm s pevnou délkou reprezentující pořadí hradel v obvodu, avšak nereprezentoval jejich vzájemné zapojení. Zde může nastat otázka zda to vadí nebo ne. Vadit by to mohlo v části křížení a mutace,kde uvedu konkrétní řešení tohoto problému.

Ohodnocení jedinců

V průběhu genetického algortimu je nutné nějak jednoznačně určit jak moc je jedinec „dobrým" řešením. Na tom stojí celé fungovaní genetických algoritmů a je proto důležité vymyslet vhodnou funkci, která převede množinu všech kladených požadavků na jedince na jednorozměrnou proměnou. Funkce má buď omezený obor hodnot a tudíž víme na kolik splňuje daný jedinec požadavky. Nebo může mít nedefinovaný obor hodnot, v tom případě dokážeme pouze posoudit který ze dvou jedinců je splňuje více.

Rozhodl jsem se implementovat verzi, kdy zhruba máme představu z hodnoty funkce, jak si jedinec vede. Funkce bude volena jako vážený součet všech slože, které budou počítány z jednotlivých kladených optimalizačních kritérií. Podrobněji v kapitole implementace 4.5.2

Selekce jedinců

Selekce jedinců, nebo-li výběr jedinců ke křížení, může probíhat několika způsoby. Většina se z nich řídí teorii Darwinovské evoluce - „nejlepší přežijí a stvoří potomky". Zde nastíním několik základních přístupů:

- turnaj z n — náhodně se zvolí n jedinců z populace a nejsilnější z nich je vybrán za vítěze, turnaj se opakuje dokud nemáme požadovaný počet jedinců
- ruletový výběr — pravděpodobnost výběru jednotlivých jedinců se volí v závislosti na jejich „síle“ v populaci, náhodně se poté vybere požadovaný počet jedinců podle jejich určené pravděpodobnosti
- další metody zaměřené na neopakujícím se výběru jedinců apod.

Nejčastěji se používá metoda turnaje jedinců, kde můžeme volbou n dobře volit požadovaný selekční tlak. Selekčním tlakem se myslí pravděpodobnost výběru silnějších jedinců. Proto jsem se rozhodl implementovat metodu turnaj ze dvou. Pro náš problém totiž nepotřebujeme velký selekční tlak. Chceme spíše zachovat rozmanitost populace, než rychlou konvergenci k lokálnímu extrému ohodnocovací funkce.

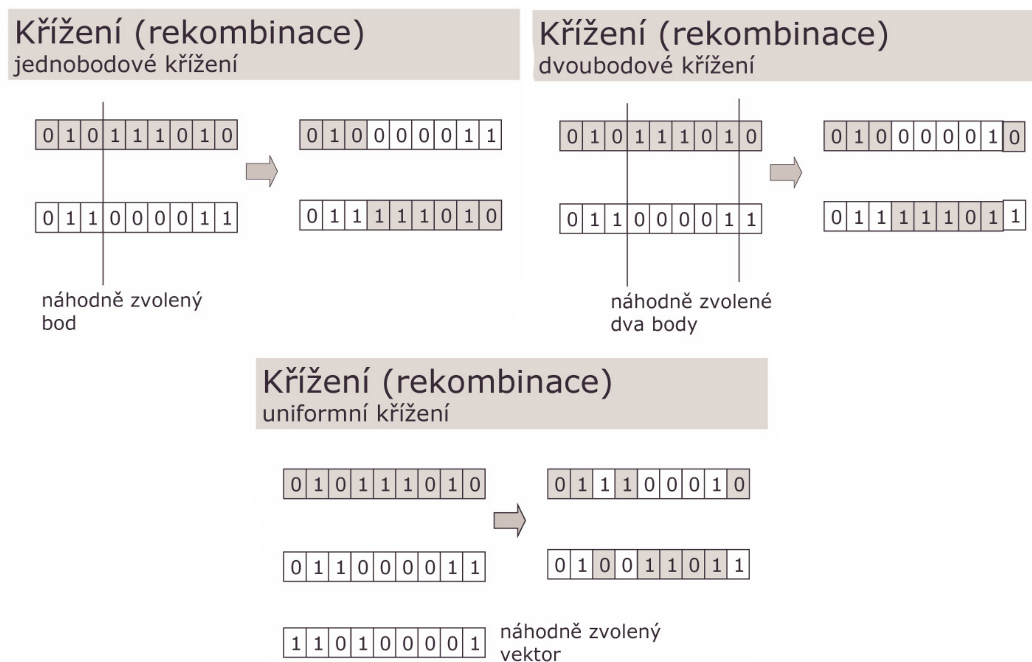
V rámci selekce dochází také k elitismu. To je metoda která přenese několik nejsilnějších jedinců do nové populace. Tímto způsobem si ponecháváme v algortimu doposud nejlepší řešení daného problému.

Křížení jedinců

Páry vybraných jedinců postupují do fáze křížení, kde se jejich geny v chromozomech vzájemně vymění. Metody křížení ovlivňují, jakým způsobem se to provede. Na obrázku 2.3 jsou graficky zobrazeny 3 základní typy křížení na binárním genetickém algoritmu. Jednotlivé bity představují geny v chromozómu. V jednobodovém křížení se oba rodiče rozdělí ve stejném místě, a prohodí si rozdělenou část chromozómu. Podobně je to i ve vícebodovém křížení. Zde se chromozóm rozdělí na n částí a rodiče si navzájem prohodí sudé části. Uniformní křížení je založeno na prohození náhodných genů. Křížením vzniknou dva noví potomci které se umístí do nové generace jedinců.

Pro náš problém bude nejlepší metoda jednobodového křížení. Poněvadž nemáme v chromozómu veškeré informace o jedinci, snažíme se udržet co nejvíce genů u sebe, abychom mohli informaci reprodukovat. Celou metodu jsme museli upravit tak, aby do potomků vložila co nejvíce nezakódovaných informací z rodičů (ohledně zapojení hradel) a zničenou část informace co nejvhodněji doplnila.

Celé to funguje tak, že se obvody obou rodičů rozpojí v náhodně zvoleném bodě a v potomcích je druhá část obvodu prohozena. Poté se snaží obě části propojit přes zbylé nezapojené vstupy a výstupy hradel. Pokud se to nepodaří, je potomek vyřazen z nové populace.



Obrázek 2.3: Různé druhy křížení

Mutace

Standartní mutace je definována jako náhodná změna genu za jiný. Je určena většinou globální pravděpodobností, zda na nově vzniklém potomkovi dojde k mutaci či nikoliv. Mechanizmus mutace zaručuje určitou rozmanitost jedinců a zabraňuje možnosti degenerace populace.

V našem případě kódování jedinců, jsme museli rošířit působnost mutace nejen na náhodnou změnu genu, ale i na náhodnou změnu zapojení hradel. Všechny přístupy které jsem implementoval jsou uvedeny v kapitole 4.5.1

Ukončují kritérium

Genetický algoritmus je iterativní a tudíž se stále snaží zlepšovat výsledek dokud nenastane ukončující kritérium. Ty jsou zpravidla 3 druhy:

- statické kritérium — je určen pevný limit po kterém se algoritmus ukončí, běžně se používá počet generací
- dynamické kritérium kvality — nejsilnější jedinec dosáhne zadané kvality řešení
- dynamické kritérium času — nejlepší řešení problému se nezmění po určitou dobu

Je výhodné implementovat alespoň jedno statické a jedno dynamické kritérium a používat obě současně. Mohla by totiž nastat situace, že dané dynamické kritérium nejde splnit, a proto se používá zároveň i statické, abychom těmto situacím předešli. Po kladném vyhodnocení ukončujících podmínek je jako výstup z algoritmu dán nejsilnější jedinec v aktuální populaci.

2.5.4 Testovatelnost

Obecně je testovatelnost definována jako schopnost obvodu být lépe (levněji,důkladněji) testován na jeho poruchy. Testovatelnost nejvíce ovlivňují dvě složky:

- řiditelnost
- pozorovatelnost

Řiditelnost spoje v obvodu je chápána jako schopnost ovlivnit jeho hodnotu pomocí různého ohodnocení vstupů. Pozorovatelností spoje se myslí schopnost zjistit jeho hodnotu na výstupech ovlivňováním pouze vstupů. Za úkol máme provést kvalitativní ohodnocení obvodu číslem, které bude určovat jeho snadnou testovatelnost resp. řiditelnost a pozorovatelnost. Obecně se dá testovatelnost rozdělit na kombinační a sekvenční. V této práci se budu věnovat jen kombinační části. Sekvenční část u metod vynecháme.

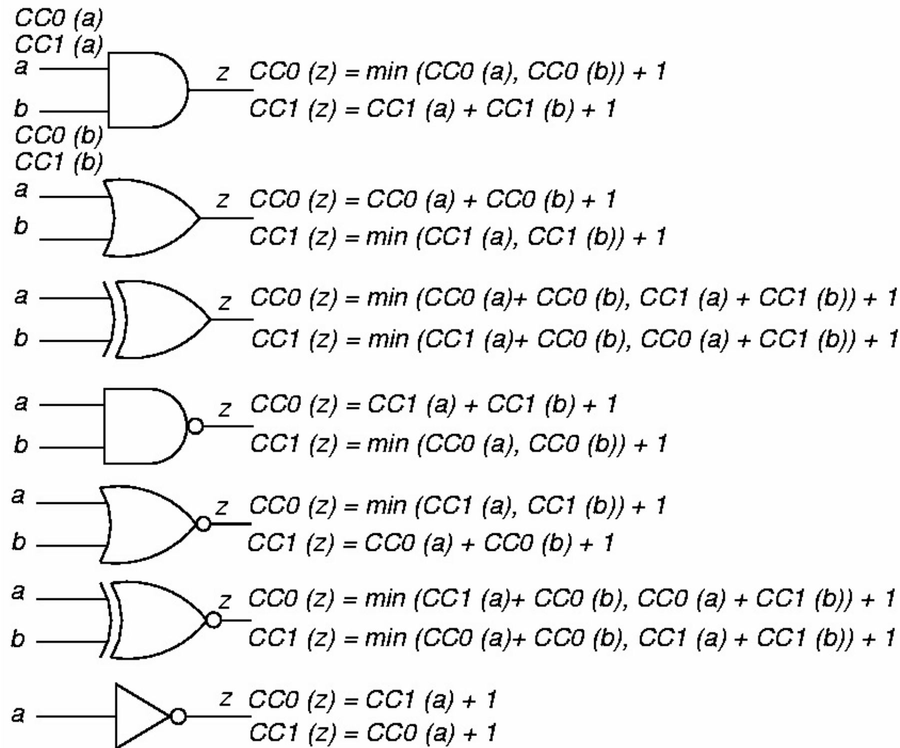
Existují metody navržené pro obvody popsané od úrovně hradel až po vyšší úrovně popisu obvodu. Nás budou zajímat ty nejpodrobnější, řešící problém na úrovni logických hradel.

Metoda SCOAP

SCOAP [8] je jedna z nejznámejších metod pro výpočet testovatelnosti. Metoda používá tzv. míry řiditelnosti a pozorovatelnosti. Ty reflektují obtížnost ovládání resp. pozorování konkrétních hodnot na vnitřních spojích obvodu. Vyšší hodnoty řiditelnosti resp. pozorovatelnosti indikují větší obtížnost této činnosti. Pro každý spoj v obvodu jsou vypočteny tři hodnoty vyjadřujících míru jeho řiditelnosti a pozorovatelnosti:

- řiditelnost nuly na spoji x — CC0(x)
- řiditelnost jedničky na spoji x — CC1(x)
- pozorovatelnost spoje x — CO(x)

Tyto míry vyjadřují kolik je zhruba potřeba nastavit spojů v obvodě, abychom změnili resp. pozorovali hodnotu na daném spoji. Řiditelnost spojů se pohybuje v rozmezí 1 (nejsnadněji) až nekonečno (nejtěžší) a Pozorovatelnost v mezích 0 (nejlehčí) až nekonečno (nejtěžší). Na obrázku 2.4 a 2.5 jsou zobrazeny výpočty pro jedntolivá hradla.

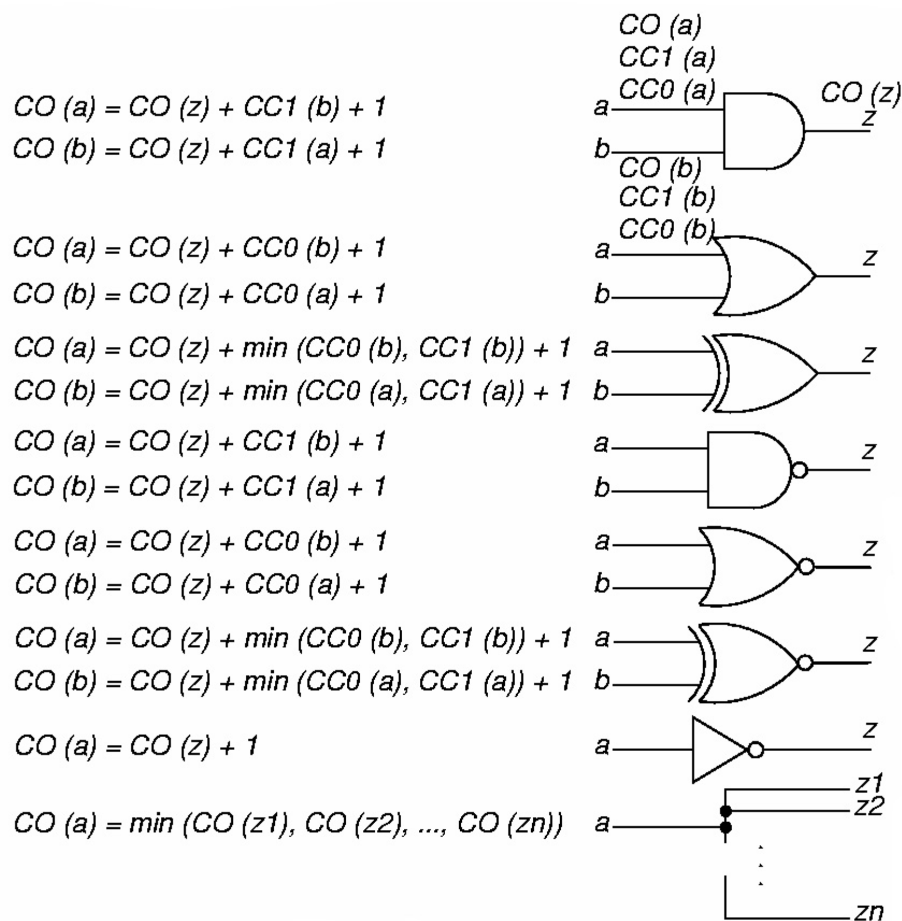


Obrázek 2.4: SCOAP: Výpočet řiditelnosti

Metoda postupuje asi takto:

- všem vstupům obvodu nastav CC0 = CC1 = 1, všem ostatním spojům CC0 = CC1 = ∞
- postupně procházej obvod od vstupů obvodu a počítej řiditelnost pomocí rovnic na obrázku 2.4
- všem výstupům obvodu nastav CO = 0, ostatním spojům CO = ∞
- postupuj od výstupů obvodu směrem ke vstupům a počítej pozorovatelnost pomocí rovnic na obrázku 2.5

Časová složitost analýzy testovatelnosti pomocí SCOAP je lineární funkcí počtu hradel, to z metody SCOAP činí atraktivní nástroj pro posuzování testovatelnosti.



Obrázek 2.5: SCOAP: Výpočet pozorovatelnosti

CAMELOT

CAMELOT [7] je metodou založenou na hradlových koeficientech. Pro každý výstup hradla je předpočítána konstanta, kterou se násobí vstupní řiditelnosti resp. výstupní pozorovatelnosti. Metoda používá 3 složky, které danému spoji x přiřazují jeho řiditelnost — CY(x), pozorovatelnost — CO(x) a testovatelnost — T(x). Všechny složky jsou v intervalu $< 0; 1 >$, kde 0 znamená, že odpovídající spoj není testovatelný (řiditelný, pozorovatelný), 1 znamená, že je testovatelný velmi jednoduše.

Jedotlivé koeficienty hradel jsou počítány takto:

- CTF(y) je koeficient řiditelnosti výstupu y a je pro každé hradlo počítán zvlášť podle pravidla

$$CTF(y) = 1 - |\frac{N(0) - N(1)}{N(0) + N(1)}|$$

  N(0) resp. N(1) jsou počty vstupních kombinací, pro které výstup y nabývá hodnoty

0 resp. 1

- OTF(x-y) je koeficient určující snadnost propagace hodnoty ze vstupu x na výstup y.

$$OTF(x-y) = \frac{N(SP : x-y)}{N(SP : x-y) + N(IP : x-y)}$$

kde N(SP:x-y) resp. N(IP:x-y) je počet citlivých nastavení vstupních kombinací kdy dochází k šíření resp. blokaci poruchy ze vstupu x na výstup y.

Jednotlivé složky CY(x), OY(x) a T(x) se počítají:

$$CY(y) = CTF(y) \times f(CY(x_1), ..., CY(x_k))$$

kde x_1 ... x_k jsou vstupy na kterých je výstup y závislý.

$$OY(x) = OTF(x-y) \times OY(y) \times g(x_1, ..., x_k)$$

kde x_1 ... x_k jsou vstupy ns kterých je závislá propagace poruchy ze vstupu x na výstup y.

$$TY(x) = CY(x) \times OY(x)$$

Algoritmus probíhá v těchto krocích.

- příprava koeficientů v každém hradle, vstupům obvodu nastaví $CY(x) = 1$ ostaním spojům na 0, výstupům obvody nastaví $OY(x) = 1$ ostatním spojům na 0
- postupujeme od vstupů k výstupům a počítáme hodnoty CY.
- postupujeme od výstupů ke vstupům a počítáme OY a TY.
- vypočteme celkové hodnoty CY, OY a TY pro obvod jako průměr příslušných hodnot ze všech spojů v obvodě.

Normalizovaná metoda

Poslední zajímavou metodu jsem objevil na internetu a je jím nedávno vyvinutá metoda ing. Strnadelem [11]. Od počátku je navrhována jako normalizovaná, tudíž kvalitativní výsledky jsou v oboru hodnot $< 0; 1 >$. Metoda je navržena na úrovni meziregistrových přenosů, ale lze ji použít i na úroveň logických hradel. Stejně jako v předchozí metodě se používají tři složky-(řiditelnost, pozorovatelnost a testovatelnost). Počítají se obdobným způsobem a to procházením obvodu postupně od vstupů k výstupům a opačně. Zde se ovšem při počítání složek uvažují veškeré cesty kterými je možné ovlivnit resp. pozorovat poruchu. Dokáže tak reflektovat nedetekovatelné poruchy a měla by tudíž podávat kvalitnější výsledek, avšak za cenu spojitou s výrazně vyšší výpočetní náročností.

Výběr metody

Implementovaná metoda, bude pravděpodobně výrazně ovlivňovat schopnost genetického algoritmu produkovat jedince se zadanou testovatelností. Rozhodl jsem se proto implementovat více metod a až z testů určit, která metoda je vhodná. Avšak z důvodů implementace jiných částí generátoru jsem se nakonec rozhodl pro metodu CAMELOT.

3 Implementace

3.1 genpla

duplicitní termy

Zde jsem pouze doimplementoval funkci, která otestuje daný term vůči všem termům v daném spojovém seznamu proti duplicitě. Šlo vlastně o porovnání dvou textových řetězců. Funkce vrací 0 v případě rozdílnosti jinak vrátí 1 a nastaví validitu termu na 1. Smazání termu a generování dalšího již bylo v práci implementováno, poněvadž jsou ostatní testy implementovány podobně.

optimalizace kódu

Z analýzy jsem určil, že největší přínos bude mít zoptimalizovat veškeré testy validity do jednoho průchodu. Proto jsem nejdříve všechny funkce, které prováděly test daného termu vůči ostatním převedl na funkce, které testují pouze dva termy vůči sobě. Poté jsem jejich kódy spojil do jedné funkce `doPostChecks(...)`,která procházela spojový seznam všech doposud vygenerovaných termů a podle zadaní používala kusy kódu původních kontrol k otestování vůči termu přidávanému.

Dále bylo nepatrně pozměněno pár částí kódu při opravách chyb.

3.2 genstate

Z analýzy nám vyšlo, že budeme pro konstrukci přechodů stavového automatu používat termy v PLA tabulce, což nám umožňuje z velké části použít 1. generátor a doplnit ho pouze o třídu, která ponese informaci o stavovém automatu jako celku. To je především počet stavů, počet ekvivalentních tříd, dále pak o minimálním zapojení automatu po splnění kontrol spojitosti apod.

3.2.1 Použité datové struktury

Pro grafové algoritmy uvedené v analýze budeme používat celkem malé množství hran vedoucích z každého uzlu grafu. Je to způsobeno tím, že i když budeme generovat úplný determinsitický automat, tak si budeme vést pouze nutné stavy pro spojitost, dostupnost a konečnost automatu. Ostatní hrany budou generovány za běhu při generování termů.

Byla proto zvolena jako nejvhodnější datová struktura reprezentující graf, metoda statického pole spojových seznamů příslušných hran o velikosti počtu uzlů. Výhodou je, že máme dostupné pohromadě všechny přechody, které vedou z jednoho uzlu.

3.2.2 Průběh algoritmu

Celý algoritmus generování probíhá zjednodušeně asi takto: Po načtení vstupních argumentů si program náhodně připraví třídy ekvivalence s počtem stavů v káždé z nich. Dále si vytvoří prázdný graf s počtem uzlů podle počtu tříd ekvivalence. Pokud jsou zapnuty příslušné kontroly, tak zajistí náhodnou spojitost grafu, pak doplní koncovost a dostupnost uzlů v grafu. Poté náhodně rozdělí zbylý počet termů mezi třídy ekvivalence. Zde přichází generátor termů z první části. Zavoláme ho postupně na všechny třídy ekvivalence a vygenerujeme tím veškeré přechody. Pak už zbývá jen vygenerovat ekvivalentní stavy z jejich tříd a podle zadání změnit některé termy a tím příslušné stavy na pseudoekvivalentní. Celý algoritmus je ještě zachycen na přiloženém obrázku a vybrané části podrobněji popsány dále.

```
\begin{figure}[h]
\begin{center}
\includegraphics[width=12cm]{alggenstate.eps}
\caption{Hrubé schéma algoritmu genstate}
\label{fig:logo}
\end{center}
\end{figure}
```

Spojitost, koncovost, dostupnost

Nakonec jsem minimální spojitost grafu řešil podle analýzy tak, že jsem si předpřipravil spojitý graf. Toho jsem docílil náhodným generováním $\sqrt{n}$ náhodně ohodnocených hran pro každé z n uzlů. Aby byl takový graf spojitý, tak první hrana každého uzlu vede v 50-ti procentech na uzel s indexem o 1 větší. Ve zbylých 50ti procentech vede na libovolný uzel s indexem vyšším, než pro který generujem hranu. Po vygenerování takto zaplněného grafu na něj pustím Jarníkův algoritmus na minimální kostru. Výsledný graf již považuji za spojitý. Pokud je zapnutá koncovost resp. dostupnost, doplním hrany tak, aby z každého uzlu vedla hrana resp. aby do každého uzlu vedla alespoň jedna hrana.

Generování přechodů

Po náhodném doplnění počtu termů pro všechny třídy ekvivalence je postupně procházíme. Pro každou třídu přednastavíme parametry generátoru a necháme si vygenerovat správný počet termů, které již budou vygenerovány podle přednastavených kontrol. Pokud generujeme úplný deterministický automat, tak je tato fáze nahrazena pseudo generátorem, který vrátí všechny možné kombinace termů. V tomto bodě máme pro danou třídu ekvivalence předgenerované termy ve správném počtu. Zavoláme na ně tedy metodu `DoplnStavy()`, která nám přímo do datové části každého termu vyplní aktuální a přechodový stav automatu. Začne vyplňovat přechody podle hran vygenerovaného grafu, zbytek přechodů doplní náhodně, s přihlédnutím na zapnutou kontrolu determinismu.

Ekvivalentní stavy

Poslední částí je převést třídy ekvivalence na ekvivalentní stavy. Celý princip je založen na tom, že stavy zařazené do stejné třídy ekvivalence můžeme libovolně zaměňovat. Jsou totiž ekvivalentní. V tomto okamžiku máme ve všech termech nastaveny přechody, které označují jednotlivé třídy. Protože víme, které stavy patří do které třídy ekvivalence, tak se celý problém zjednodušil na pouhé nahrazování tříd v přechodech za náhodný stav z příslušné třídy. Toho docílíme tím, že každému stavu kopírujeme termy z příslušné třídy ekvivalence a přepisujeme indexy tříd na indexy stavů podle předem vygenerované tabulky.

Pseudo-ekvivalentní stavy jsou generovány jako procentuální množství termů, které nebudou mít definován určitý stav do kterého přechazejí. Procházejí se postupně všechny termy a se zadanou pravděpodobností se změní přechod na nedefinovaný.

4 Implementace gennetwork

Tento generátor používal na generování obvodu úplně jiný přístup, než oba dva předchozí, proto bylo nutné navrhnout ho kompletně od začátku. Byl mnohem náročnější na implementaci, proto se zde o něm rozepíši trochu více. Především proberu algoritmus a datové struktury trochu více do hloubky.

4.1 Datové struktury

Z analýzy problému bylo jasné, že si budeme muset v paměti uchovávat celý obvod v jediné struktuře, kvůli použití genetického algoritmu. Dále bylo jasné, že si budeme muset vést celé zapojení jednoho obvodu v podobě grafu nad kterým budou definovány elementární metody na jeho úpravu. Tyto metody musí udržovat integritu dat zvolené formy grafu.

4.1.1 Graf zapojení obvodu

Při implementaci grafu bylo asi ze všeho nejdůležitější rozhodnout, jestli budou spoje obvodu jako samostatné entity, nebo budou vedeny jen jako ukazatele mezi uzly grafu. S tím také úzce souviselo, jak bude reprezentováno větvení hran. Uvedu zde nástin pro a proti u obou případů.

spoj jako samostatná entita
klady:

- veškeré informace si nese sám. Jestli jde o globální vstup, do jakých hradel je zapojen, kolikrát je větven, počáteční a koncová úroveň v obvodu
- máme přistup k seznamu všech hradel, takže operace nad všemi nebo nad různě vymezenou částí je jednodušší
- spoj má implementované svoje elementární metody

zápory:

- zabírá více místa v paměti
- podstatná část informací, kterou si spoj vede, je redundantní a tudíž je zde potencionální problém nekonzistence dat, včetně redundance metod spoje

spoj jako ukazatel na následníka
klady:

- rychlejší procházení celým obvodem
- nevedení si zbytečných redundantních informací
- veškeré operace mohou být implementovány jako metody uzlu grafu — méňě metod

zápory:

- pro obvod musí být vytvořen jeden velký uzel(hradlo) s globálními vtupy a výstupy
- všechny informace o spoji si musíme pokaždé dopočítat

Nakonec jsem zvolil spoj jako samostatnou entitu kvůli přednosti, že si všechny informace nese sám spoj a nemusejí se dopočítávat. Tím jsem mohl řešit další problém a to jak implementovat větvení spojů. V úvahu připadali 3 možnosti:

1. na každý výstup hradla lze připojit více spojů — budeme mít jen větvené spoje, větvení se pozná z parametrů hradla
2. vytvoříme nový typ hradla - větvení, informaci o větvení ponese toto hradlo a spoj, kterej do něj vede
3. každému spoji jde přiřadit více koncových hradel

Všechny tři řešení měla svá pro a proti. Kvůli dodržení jednoduchosti spojů a uzlů v grafu jsem povolil vždy zapojovat na jejich přípojné body pouze jednu entitu. Proto jsem zvolil implementaci č.2. Ostatní implementace měli problémy například při větvení globálního vstupu nebo pokud jedna větev byla globálním výstupem. Zároveň jsem si však uvědomoval, že zmenšování či zvětšování větvení spoje bude v této implementaci mnohem složitější, než ve zbylých dvou.

Jelikož v obvodě nesmějí být zpětné vazby, zavedeme si vlastnost úroveň. Všechny hradla na jisté úrovni smějí být na výstupu spojeny jen s hradly na úrovni vyšší. Analogicky všechna hradla na jisté úrovni smějí být na vstupu spojeny pouze s hradly, které jsou na nižší úrovni. Zároveň jsem zvolil, že všechna hradla na větvení budou vždy jen o jednu úroveň výš než hradlo, ze kterého vede větvený spoj. Aby se nám nepletla obyčejná hradla s hradly na větvení, tak jsem určil, že obyčejná hradla budou umisťovaná pouze do lichých úrovní. Do sudých úrovní budeme umisťovat pouze hradla větvení. Tím jsme dosáhli toho, že můžeme hranu později větvit, i když vede mezi sousedními obyčejnými hradly. Každé nově generované hradlo, přidávané do obvodu, budem generovat na nejvyšší neobsazené úrovni. Tím dosáheneme co největší rozmanitosti generovaných obvodů.

4.1.2 Entity

`class Thradlo`

Tato třída představuje v celém programu jedno hradlo. Mezi její nejdůležitější vlastnosti patří: `typ`, `pocet_vstupu`, `pocet_vystupu`, `zapojene_vstupy`, `zapojene_vystupy`, `nazev`, `uid`, `uroven`, `realna_uroven`, `obvod`, `next`, `vstupy`, `vystupy`. Většina vlastností je zřejmá podle jejich názvů. Vlastnost `next` je ukazatel na další hradlo v obvodě a implementuje tak jednoduchý spojový seznam všech hradel. Vlastnost `vstupy` resp. `vystupy` je pole ukazatelů na připojené spoje na vstupu resp. výstupu.

Metody jsou zaměřeny především na připojování a odpojování spojů od vstupů a výstupů hradla. Metoda `zmensi_hradlo()` a `zvetsi_hradlo()` se už podle názvu orientují na změnu počtu vstupů/výstupů v hradle a používají se především u hradel na větvení spojů. Dále jsou implementovány metody na výpočet různých koeficientů a testovatelnosti. Některé metody rozeberu později podrobněji.

`class Tspoj`

Třída implementující spoj mezi hradly. Její důležité vlastnosti jsou: `uid`, `vstup`, `vystup`, `zac`, `kon`, `vetveni`, `op`, `fc`, `obvod`, `next`. Vlastnost `vstup` a `vystup` specifikuje, jestli jde o globální vstup nebo výstup v obvodě, `zac` a `kon` jsou ukazatele na připojená hradla. Jestli je spoj operační[1] určuje vlastnost `op` a `fc` určuje jestli je funkční[2]. Tyto vlastnosti pomáhají při kontrole větvení a pomáhají při kontrole konzistence dat. Vlastnost `next` jako v případě hradla tvoří jednoduchý spojový seznam všech spojů v obvodě.

Implementované metody jsou převážně na připojení a odpojení ke vstupům hradel, případně převedení spoje na globální vstup/výstup, dále pak na počítání testovatelnosti a nakonec zajímavá metoda `Rozvetvi()`, která obstarává celkem složité větvení spojů. O té se zmíním ještě později.

`class Tobvod`

Tato třída jak již název napovídá představuje jeden celý kombinační obvod. Jako vlastnosti má veškeré zadané parametry generovaného obvodu např. `pocet_vstupu`, `maximalni_vetveni`, `pocet_hradel`. Dále obsahuje některé pomocné věci při generování triviálního obvodu. Pro představu `cislovani_hradel`, `akt_vstup`, `max_uroven`. Další nedílnou součástí jsou parametry určující pravděpodobnost výskytu jednotlivých

[1] Operační spoj jsem si zavedl pro spoj, který je globální vstup a nebo je připojen na výstup nějakého funkčního hradla

[2] Funkční spoj označuje spoj který je přiveden na vstup funkčního hradla nebo je to globální výstup

hradel, stejně jako jejich maximální fan-in a pravděpodobnosti na jejich vícevstupové modely. Nesmíme zapomenout na ukazatele na seznam spoju a hradel `Hradla`, `Spoje` a ještě na obousměrně zřetězený spojový seznam hradel seřazených podle úrovně — `seznam_zac`, `seznam_kon`. Statistické informace o obvodu nese třída uložena v parametru `info`. Mezi další důležité vlastnosti patří pole vah pro výpočet fitness funkce, stejně jako globální vlastnosti obvodu. Do toho počítaje pro příklad `celkova_testabilita` nebo `realne_urovne`.

Mezi duležité metody třídy patří například: `vygeneruj_obvod`, `vytvor_nove_hradlo(..)`, `spocti_urovne()`, `spocti_fitness()`, `proved_mutaci(..)`. Dále pak třída deklaruje metody na uložení v různých formátech a jiné pomocné metody. O některých se budu zmiňovat ještě později.

`class Tinfo`

V této třídě jsou uchovány statistické hodnoty již vytvořeného obvodu. Používá se pro počítání fitness funkce při zapnuté kontrole počtu hradel v obvodě. Tyto informace se ještě použijí při výpisu informací do ukládaného formátu a na obrazovku.

`class Thlavni`

Tato třída je nadevšemi třídami a nese informace o celém algoritmu, především pak struktury pro správné použití genetického algoritmu. Je to především pole ukazatalů na obvody vystihující aktualní populaci jedinců — `generace`.

Třída má jen 2 metody a to jsou `zaloz_generaci()` a `nova_generace()`. První z nich generuje počáteční populaci pomocí metody `Tobvod::vygeneruj_obvod` a druhá z nich implementuje genetický algoritmus. Ten popíšu později.

4.2 Průběh algoritmu

Celý průběh generování sítě logických hradel probíhá zhruba asi takto: Nejprve se načtou vstupní parametry z příkazové řádky, poté se předpřipraví datové struktury a přepočítají se pravděpodobnosti generování různých typů hradel. Pokud není zapnuté ani jedno optimalizační kritérium, tak se vygeneruje jeden triviální obvod, uloží se a program končí. V opačném případě se vytvoří první generace pomocí metody

`Thlavni::zaloz_generaci()`. Následovně se začne populace jedinců zušlechťovat pomocí metody `Thalvni::nova_generace()`. Program poté zušlechťování iterativně opakuje, dokud se nesplní jedno ze dvou možných ukončujících kritérií :

1. počet prošlých generací se rovná zadanému maximálnímu počtu generací
2. počet generací při kterých nebyl změněn nejsilnější jedinec se rovná hodnotě zadané v příkazové řádce. Jinak se použije standratní hodnota rovna 50

Po ukončení iterativní smyčky máme na počátku populace nejlepšího jedince, toho uložíme do zvolených formátů a program končí.

4.3 Popis některých důležitých metod

4.3.1 Základní operace se spoji a hradly

U všech těchto metod bylo velmi důležité striktně dodržet konzistenci dat. Zde se projevil výběr datových struktur. Bylo nepodstatně těžší tyto metody bezchybně naprogramovat, než kdybych zvolil struktury s méně redundantními daty.

Nejsložitější případy vznikali při rozvětvování ještě nevětveného spoje a naopak při zmenšování větvícího hradla. Pokud se totiž hradlo zmenšilo až na jeden výstup, tak se zároveň samo smazalo a vstupní s výstupním spojem se sloučili v jeden. Tím vznikal potencionální problém pokud jsme měli někde ukazatel na spoj, který se při tomto zmenšování smazal. Na to si bylo potřeba dávat velký pozor.

4.3.2 Metoda na tvorbu nového hradla

`Tobvod::vytvor_nove_hradlo(..)`

Tato metoda má argument, kterým se vybírá jestli generujeme i hradlo NOT nebo ne. Při generování triviálního obvodu bylo totiž potřeba generovat pouze hradla s více vstupy, abychom mohli snížit v celém obvodě nezapojené výstupy hradel. Metoda generuje hradlo typu i ze všech typů hradel n s pravděpodobností $P(i)$, která se počítá ze zadaných celočíselných indexů výskytů všech typů hradel $p(j)$ a to podle tohoto vzorce

$$P(i) = \frac{p(i)}{\sum_{j=1}^{n} p(j)}$$

Indexy výskytů hradel se dají měnit z příkazové řádky programu, standartně jsou nastaveny na hodnotu 5.

Podle podobného principu se počítá i pravděpodobnost generování vícevstupového hradla v rámci jednoho typu hradla. Standartně jsou všechna hradla nastavena na maximální fan-in. Toto se dá předefinovat z příkazového řádku.

Všechny zde uvedené proměnné jsou v rámci jednoho zvoleného typu hradla. Pokud si označíme i jako počet vstupů generovaného hradla, n maximální fan-in, tak index výskytu hradla $p(i)$ je počítán jako

$$p(i) = \frac{n-1}{i-1}$$

Pravděpodobnost $P(i)$ generování hradla s fan-in rovno i je potom

$$P(i) = \frac{p(i)}{\sum_{j=1}^{n} p(j)}$$

Tato pravděpodobnost je v průběhu celého programu neměnná a nedá se nijak ovlivnit.

Metoda potom pomocí konstruktoru vytvoří hradlo s vypočítanými parametry.

4.4 Generování triviálního obvodu

`Tobvod::vygeneruj_obvod()`

Podle analýzy problému jsme určili, že budeme generovat triviální obvod s pevně danými neměnnými parametry a toho jsem se při implementaci musel držet. Šlo především o počet vstupů, počet výstupů, počet hradel a maximální větvení. Dále jsme se měli držet pravděpodobnosti generování typů hradel. To už jsme vyřešili předchozí popisovanou metodou.

Metoda si vede 2 pomocné pole ve kterých si ukládá:

- spoje, které můžeme alespoň jednou rozvětvit a nepřesáhneme zvolené maximální větvení — pole SP
- nezapojené výstupy operačních hradel[3] — pole HR

Metoda potom vygeneruje hradlo a pokud existuje nezapojený globální vstup tak ho s 90-ti procentní pravděpodobností připojí na 1. vstup hradla. Ve zbylých 10-ti procentech zkouší buď větvit spoj z pole SP, nebo připojit na hradlo z pole HR. Zbylé vstupy se

[3]Nazývám tak všechny typy hradel mimo hradla větvení.

pokouší zapojit s těmito pravděpodobnostmi — 60% globální vstup, 30% připojit na výstup náhodného hradla z HR, 30% větvení náhodného spoje z SP. Pokud se nám z nějakého důvodu nepodaří zapojit nějaký vstup, zkusíme zmenšit počet vstupů hradla, nebo ho smažeme a zkusíme vygenerovat znovu.

Při každém nově vytvořeném spoji, a pokud není omezeno větvení na 1, tak spoj přidáme do pole SP. Při úspěšném zapojení celého hradla ho přidáme do pole HR. Zároveň musíme při větvení koukat, jestli nepřesáhneme maximální větvení, v tom případě musíme hranu z pole SP odstranit.

Když máme zapojené všechny globální vstupy, tak si náhodně zvolíme počet globálních výstupů, které budou připojeny přímo na výstup z operačního hradla. Volíme mezi 70-ti a 90-ti procenty ze všech globálních výstupů.

Následně vygenerujeme nové hradlo a vypočítáme pro něj 2 hodnoty, které určují kolik jeho vstupů musíme minimálně a maximálně zapojit přímo na výstupy předešlých hradel z pole HR. Zároveň volíme pravděpodobnosti zapojení na hradlo z HR nebo větvení nějakého pole z SP tak, abychom v průměru měli jeden vstup zapojený na HR. Tím dosáhneme toho, že budeme počet hradel v poli HR stále udržovat na přibližně podobné hodnotě, a s blížícím se koncem generovaných hradel do správného počtu se budeme blížit i zvolenému počtu přímo zapojených globálních výstupů. V této fázi dodržujeme stejná pravidla přidávání a odebírání hradel a spojů z pomocných polí SP a HR.

Po dokončení této fáze nám stačí zapojit výstupy hradel z pole HR na globální výstupy a zbylé nezapojené globální výstupy zapojit na nějaké větvení z pole SP.

Chci připomenout, že pokaždé při pokusu o rozvětvení spoje z pole SP, nejdříve kontrolujeme, jestli je možno připojit daný spoj na hradlo pomocí metody `Thradlo::test_pripojeni_vstupu()`. Metoda kontroluje, aby na jedno hradlo nebyly připojeny 2 větve ze stejného spoje. Kdyby toto nastalo, logická fuknce hradla by se v mnoha případech degradovala.

Tím máme vygenerován triviální náhodný obvod, který splňuje požadované parametry.

Pokud se nějakým důvodem nepovede zapojit naponěkolikáté vygenerované hradlo, tak se celý obvod smaže a funkce vrátí hodnotu odlišnou od nuly.

4.5 Genetický algoritmus

Ve zvolené implementaci obvodu není nutné nikterak kódovat jedince pro účely genetického algoritmu. Pro náš algortimus potřebujeme chromozóm s pevnou délkou, který by jistým způsobem popisoval celý obvod. Všechny generované obovdy mají stejný počet

operačních hradel a každé je na svojí úrovni. Proto můžeme brát hradla jako geny a spojový seznam hradel jako chromozóm. Způsob jak se vypořádává algortimus se spoji je vysvětlen v části o křížení.

Na začátku genetického algoritmu, se připraví počáteční generace cyklickým voláním metody na vygenerování triviálního obvodu.

`Thlavni::nova_generace()`

Tato metoda implementuje všechny fáze genetického algoritmu mimo mutace. Na začátku se vytvoří nová prázdná populace, a podle principu elitismu se do ní začnou kopírovat nejsilnější jedinci z generace předchozí. V našem případě je elitismus podle analýzy nastaven na 1, tudíž se kopíruje 1 jedinec. Dále jsou do nové populace vygenerováni noví trivialní jedinci, a to v počtu, který se rovná 8-mi procentům velikosti populace. Tu jsme zvolili jako turnaj z n. V našem případě je zvolena konstanta 2. Větší konstantou bychom docílili většího selekčního tlaku genetického algoritmu, a tím bychom více směřovali k hledání maxim v okolí již známých řešení. Avšak v našem případě je stavový prostor problému velmi složitý a má jistě mnoho lokálních extrémů. Proto volíme spíše randomizovaný výběr. Je to vidět i na trošku volnější metodě křížení, která v sobě má také jistou náhodnost.

Když máme vybrané rodiče, tak dochází ke křížení. Poněvadž máme ve všech obvodech stejný počet hradel a každé hradlo je ve svoji úrovni obvodu, je velice vhodné křížit dva obvody jejich rozdělením za určitým hradlem. Problém může nastat jedině při dopojování spojů mezi hradly. Celý mechanizmus funguje asi takto:

Nejprve se náhodně vybere hraniční hradlo a to zhruba mezi 10-ti a 90-ti procenty obvodu — aby bylo co napojovat. Nový obvod se vytvoří překopírováním 1. části hradel z 1. rodiče a zbylý počet hradel z 2. rodiče. Poté se zapojí hradla v první části podle 1. rodiče. Spoje které vedou do 2. části se nekopírují. Analogicky se to udělá s 2. částí obvodu. Zde se nebudou kopírovat spoje, které vedou z první části obvodu.

Nyní si algoritmus připraví 5 pomocných polí $hr1$ až $hr4$ a sp_pole. Jejich význam je následující:

1. vede všechny volné výstupy operačních hradel v 1. části
2. vede všechny voné výstupy hradel větvení v 1. části
3. vede všechny volne vstupy operačních hradel v 2. části
4. vede všechny hradla větvení v 2. části, v poli je ma uloženy $n-1$ krát, kde n je počet výstupů daného hradla

5. vede si všechny možné spoje k větvení v 1. části

Algoritmus se teď pokouší dozapojit chybějící spoje pomocí těchto 4 polí. Snaží se dosáhnout, aby byly všechny globální vstupy a výstupy zapojené, stejně jako aby zůstala pole $hr1$ a $hr3$ prázdná. Pokud algoritmu dojdou vstupy hradel v $hr3$ a globální výstupy, snaží se je nahradit zmenšením větvení v druhé části obvodu (pole $hr4$). Pokud mu naopak dojdou v první části globální vstupy a výstupy hradel z pole $hr1$, snaží se tyto chybějící spoje zapojit již na původně obsazené výstupy větvících hradel z pole $hr2$. Dojdou-li i ty, snaží se větvit spoje z pole *sp_pole*. Pokud se mu podaří vyčerpat pole $hr1$ i $hr3$ máme všechno zapojené a stačí nám zmenšit všechna nezapojená větvení z pole $hr2$. V opačném případě mažeme celý obvod a přecházíme k tvorbě nového potomka rodičů.

Po úspěšném křížení je s 5-ti procentní pravděpodobností udělána mutace obvodu. Následně jsou potomkovi dopočítány všechny kvalitativní vlastnosti, včetně fitness funkce a je zařazen do nově vznikající populace. V tomto okamžiku je potomek připraven a dalším cyklem se tvoří další potomek s drobným rozdílem. Pokud rodiče byli vybrání v minulé iteraci, jsou vybráni znovu, ale opačně, takže dojde k opačnému křížení. Tvorba potomků pokračuje, dokud není zaplněna celá populace.

Nakonec se celá nová populace seřadí podle výsledku fitness funkce a stará populace se nahradí novou.

4.5.1 Mutace obvodu

`Tobvod::proved_mutaci()`

Metoda je volána na správně zapojený obvod a provede náhodně jednu ze tří druhů mutací:

- zamění 1 hradlo za jiné generované podle platných pravděpodobností
- vybere náhodný spoj, k němu najde náhodný vhodný jiný spoj a dojde k prohození zapojení konců mezi oběma spoji
- náhodně vybere 1 větvení a přepojí jeho část spojů na jiné větvení

.

Nejprve si pro všechny typy mutací připravíme 2 pomocná pole.

- pole všech spojů které se můžou větvit, seřazené podle úrovně počátku v obvodu. Pole SP
- pole všech hradel větvení v obvodě. Pole HR

Chci podotknout, že ve všech zde uvedených metodách kontroluji, při zapojování větvené spoje na libovolné hradlo, aby nedošlo k zapojení dvou různých větví stejného spoje.

Záměna hradla

Algoritmus vybere jedno náhodné operační hradlo z celého obvodu, které bude měnit za jiné. Poté si pomocí pole HR vypočítá, kolik musí mít nové hradlo minimálně a maximálně vstupů, aby šlo zapojit. Voláním metody `obvod::vytvor_nove_hradlo()` se vytvoří hradlo, které splňuje podmínky na počet vstupů. Začnou se přepojovat spoje ze starého hradla na nové. S výtupem problém není, ten mají obě dvě hradla pouze jeden. U vstupů je situace jiná. Nejprve se přepojí všechny operační spoje[4], pak i ostatní. V případě, že nové hradlo má ještě nějaké nezapojené vstupy, ty se vhodným větvením doplní. Pokud po úplném zapojení nového hradla zůstaly některé spoje na starém hradle, tak buď zmenšíme větvení ze kterého spoj vychází, nebo zmenšíme náhodné větvení na vyšší úrovni. Tím nám vznikne neobsazený vstup na hradle, kam již můžeme spoj ze starého hradla přepojit.

Překřížení dvou spojů

Algoritmus vybere náhodně libovolný spoj z celého obvodu, poté náhodně vybírá jiný spoj se kterým by si vyměnil svůj koncový prvek. Postupně zkouší náhodný spoj a kontroluje (dle řady omezujících podmínek) jestli je možné si konce vyměnit:

- kontorla proti zpětné vazbě — Začátky obou spojů musí být připojeny na nižší úroveň než menší úroveň obou konců
- nesmí vzniknout spoj mezi hradly větvení, kontrolujeme vlastnosti op a fc spojů.
- nesmí se přímo propojit globální vstup a výstup jedním spojem
- pokud přepojujeme větvený spoj, tak ho nesmíme přepojit na hradlo, do kterého už vede jiná větev téhož spoje

Pokud spoj všechny podmínky splní, tak si v pořádku mohou oba spoje vyměnit konce. V opačném případě se spoj vyřadí z výběru a zkusí se náhodně jiný. Kdyby nastala situace, že nenajdeme žádný vhodný spoj, tak výměna konců neproběhne.

[4]spoje co mají nastavenu vlastnost op. Neboli spoje co nevedou z větvení.

Přesouvání některých větví

Náhodně vybereme z pole SP jedno větvení ze kterého budeme přesunovat větve jinam. Poté náhodně zvolíme počet větví, které budeme přepojovat. Na původním hradle nám musí zbýt alespoň jeden spoj. Začínáme s odpojováním zvolených spojů od větvení. Postupujeme postupně, spoj odpojíme od větvení, hradlo ihned zmenšíme, spoj odpojíme také od koncového hradla a smažeme ho. Zbývá nám zapojit nově vzniklý nezapojený vstup hradla rozvětvením vhodného Spoje, které máme seřazené v poli SP. Pokud se nám to nepodaří, vždy můžem zapojit zpátky na původní hradlo větvení. Takhle postupujeme se všemi zvolenými spoji k přesunu.

4.5.2 Fitness funkce

Fitness funkce se počítá z několika složek, které zvolíme při startu algoritmu. Všechny složky jsou implmenentovány jako reálné číslo v intervalu $< 0; 1 >$. Toto číslo reprezentuje jak moc obvod splňuje příslušné optimalizační kritérium a do výsledné fitness funkce je promítnuto přes svou váhu. Standartně jsou váhy pro aktivované složky nastaveny na 1.0. Kritéria které jsme nevybrali, mají váhu nastavenou na 0 a tím se do výsledku nepromítnou. Celý program umožňuje tyto váhy libovolně přenastavit a tím vlastně kvalitativně určit, která kritéria jsou pro nás jak moc důležitá. Program implementuje 5 různých kritérií optimalizace.

1. počet úrovní logicého obvodu. Tzn. počet hradel přes které vede nejdelší cesta z globálního vstupu na globální výstup.

2. správnost počtu všech typů generovaných hradel. Značí, že chceme co nejvíce dodržet zadané pravděpodobnosti generovaných hradel.

3. řiditelnost obvodu. Chceme optimalizovat obvod na požadovanou úroveň řiditelnosti

4. pozorovatelnost obvodu. Chceme optimalizovat obvod na požadovanou úroveň pozorovatelnosti

5. testovatelnost obvodu. Chceme optimalizovat obvod na požadovanou úroveň testovatelnosti

Dále je do fitness funkce promítnuto malé zvýhodnění potomků, kteří přežili z dřívější populace. Je to z důvodu situace, kdy více generovaných různých potomků má stejné maximální ohodnocení použitých složek. Poněvadž používám k řazení potomků standartně

implementovanou funkci `qsort()`, která není stabilní[5], tak by se s velkou pravděpodobností jedinci na začátku populace stále obměňovali. To nám vadí z důvodu, že konec genetického algoritmu je závislý na počtu generací, kdy je nejsilnější jedinec nezměněn.

Fitness funkce fit konkrétního obvodu je tedy počítána podle vzorce, kde n je počet složek, $f(i)$ je hodnota složky i, w_i je váha složky i a s je stáří tohoto obvodu:

$$fit = \sum_{i=1}^{n} [w_i \times f(i)] + 0.00001 \times sig(s)$$

Výpočty jednotlivých složek

Úroveň — Když r je úroveň obvodu, n je zadaná úroveň, tak složka fitness funkce f se počítá:

$$f = \frac{1}{|n - u| + 1}$$

Počet hradel jednotlivých typů — Složka fitness funkce f je počítána podle vzorce, kde N je počet typů hradel, n_i je počet hradel v obvodě typu i, $P(i)$ je zadaná pravděpodobnost hradla typu i v obvodu, m je počet hradel v obvodě.

$$f = 1 - \frac{4 \times \sum_{i=1}^{N}[|m \times P(i) - n_i]}{2 \times m}$$

Testovatelnost — Odpovídající část fitness funkce f, kde t je vypočítaná testovatelnost z obvodu, zt je zadaná testovatelnost.

$$f = 1 - |zt - t|$$

Obdobně se počítají i složky řiditelnosti a pozorovatelnosti.

4.5.3 Ostatní parametry

Velikost populace

Další důležitý parametr genetického algoritmu, které jsme doposud nezmínili, je především velikost populace. Ta ovlivňuje rozmanitost a počet bodů ve stavovém prostoru ze kterých vycházíme. Vysokým nastavením této hodnoty zbytečně bez velkého přínosu

[5]Nestabilní metoda řazení, nám nezaručí ve výsledku stejné pořadí prvků se shodným klíčem, jaké bylo před řazením

zvýšíme časovou náročnost každé generace. Ovšem moc malé nastavení způsobí již zmiňovanou malou rozmanitost jedinců, tím zvyšujeme šanci na degeneraci populace a zvětšujeme šanci uvíznout v lokálních extrémech ohodnocující funkce. Ze zkušeností jsem počáteční hodnotu nastavil na 50 jedinců. Avšak po přidání funkce nových náhodných jedinců v populaci a po experimentálním testování, jsem zvýšil standartní velikost populace na 80. Toto testování je uvedene v další kapitole. Zároveň jsem doimplementoval možnost měnit velikost přímo z příkazové řádky.

Pravděpodobnost mutace

Podle [4] je vhodné nastavit mutaci na pravděpodobnost někde kolem 5ti procent. To se mi i osvědčilo u jiných dříve řešených problémů. Přesto jsem udělal pár testů na malém počtu dat, kde se nepotvrdil nijak pozorovatelný vliv mutace na výsledcích algortimu. Pravděpodobně to bude tím, že již námi zvolený způsob křížení v sobě nese jistou náhodnou změnu obvodu.

Ukončující kritérium

V programu jsou implementována dvě odlišná ukončující kritéria. První je statické a program ukončí po projití pevně stanoveného počtu generací. Toto kritérium nemá standartní hodnotu a jestli jej chceme použít, musíme ji zvolit na příkazovém řádku. Standartně je definováno druhé dynamické ukončující kritérium. To daný algoritmus a program ukončí, pokud se v posledních n generacích nezměnil nejsilěnjší jedinec. Standartně je definováno na hodnotu 50.

4.6 Testovatelnost

Implementace výpočtu celkové testovatelnosti obvodu do generátoru byl klíčový. Kvůli tomuto požadavku jsme především zvolili celou metodu iterativních algoritmů pro generaci obvodu, abychom mohli zpětně ovlivňovat zapojení obvodu a tím testovatelnost přizpůsobovat požadavkům.

Poněvadž je výpočet testovatelnosti v obvodu sám o sobě docela obtížný úkol, rozhodl jsem se použít již některou dříve vymyšlenou metodu bez úprav. Především byl problém ve zvolení správné testovatelnosti. Nároky byli zhruba tyto:

- schopnost počítat testovatelnost v krátkém čase s co nejmenší složitostí
- postačuje nám kombinační testovatelnost
- výsledek výpočtu testovatelnosti musíme být schopni rozumně převést na interval $< 0; 1 >$

Výběr z metod byl složitý. Nejvíce se mi líbila metoda SCOAP, avšak zde byl vážný problém s objektivním převodem celočíselných výsledků do intervalu $< 0; 1 >$, což byla nutnost při námi navrženém způsobu výpočtu fitness funkce. Přesto je tato metoda v kódu programu implementována, ale není používána.

Další zajímavá metoda byla metoda ing. Strnadela. Způsob jejího výpočtu byl založen na počítání všech cest v obvodu. Složitost jejího výpočtu je tedy závislá na velikosti obvodu a jeho větvení. Což neodpovídá našim požadavkům.

Proto jsem z metod vysvětlených v analýze, vybral podle požadavků metodu CAMELOT. Měl jsem však obavy, že metoda, která používá plošné koeficienty, nebude tak objektivní. Při větších obvodech nemusí podávat správné výsledky kvůli četnému větvení spojů, či redundanci celých částí obvodu. Tuto domněnku bylo potřeba v řádně otestovat.

Její implementace je v souladu s jejím popisem v kapitole 2.5.4. Poněvadž jsou se u jednoduchých hradel podílejí všechny vstupy na zcitlivění cest stejným podílem, tak v implementaci jsou za funkce f(...) a g(...) zvoleny aritmetické průměry.

Algoritmus probíhá ve dvou průchodech. V prvním jdeme postupně od vstupů k výstupům přes hradla seřazená podle hloubky a počítáme řiditelnost. V druhém průchodu jdeme přes všechny hradla od výstupů a počítáme pozorovatelnost společně s testovatelností. Celková řiditelnost, pozorovatelnost a testovatelnost se pak výpočítají jako průměr ze všech spojů v obvodě.

5 Testování

Pro nastavení správných parametrů použitých algortimů v generátoru a pro ověření správného fungování, bylo s přihlédnutím na požadavky vytvořeno několik testů.

1. test na určení velikosti populace genetického algoritmu
2. test, který zjišťuje jak ovlivňuje nastavená testovatelnost obvodu jeho nedetekovatelné poruchy a resyntézu.
3. test na zjištění, jak ovlivňuje velikost generovaného obvodu jeho nedetekovatelné poruchy a schopnost resyntézy.
4. porovnání výsledku testovatelnosti obvodů z množiny ISCAS s vygenerovanými obvody podobných parametrů

5.1 Nastavení testů

Pokud není v testech uvedeno jinak, tak všechny testy probíhali se standartně nastavenými parametry generátoru. Probíhali na stroji s konfigurací : Pentium Core2-duo 2Ghz, 2GB RAM, Microsoft Windows XP Professional.

Testovatelnost

Ke zjišťování reálné testovatelnosti byl použit nástroj Atalanta vytvořena na Technické univeritě ve Virginii. Konkrétně upravený model Atalanta-M 1.1 panem Ing. Fišerem. Podrobný popis, návod a možnost stažení najdete v [1].

V testech jsme tímto programem v jeho automatickém módu zjišťovali kolik se v obvodě nachází celkem poruch — označujeme "faults", kolik poruch je zaručeně nedetekovatelných — "r_faults" a kolik poruch se detekovalo — "d_faults".

Dále jsme se zkusili v jeho simulačním módu otestovat reálnou testovatelnost. Pomocí funkce generátoru náhodných vektorů, jsme vygenerovali pevný počet vstupních vektorů. A přes simulaci jsme zjišťovali kolik poruch tato metoda neobjeví, oproti automatickému módu. Detekované poruchy náhodnými vektory jsem označil "RPDF". Sloupeček označený "nr_faults" označuje poruchy obvodu, které nebyly určeny jako redundantní. Je to vlastně kolonka "faults"-"r_faults".

Resyntéza obvodu

K resyntéze obvodu byl použit nástroj SIS. Možnost stažení i s podrobným návodem je v [5] resp. v [3]. Zde bylo poměrně těžké určit a vypočítat hodnotu, která by podávala vypovídající hodnotu o možnosti minimalizaci obvodu. Proto jsem zavedl hodnotu "influence", které pro každý výstup obvodu znamená, na kolika vstupech obvodu je závislý. Pokud má obvod více výstupů celková "influence" se vypočítá jako průměr každého výstupu. Toto číslo má vypovídat o schopnosti generovat obvody, které budou citlivé na vstup a tímpádem lépe testovatelné.

Výpočet "influence" jsem prováděl programem SIS 1.2. Minimalizace obvodu probíhala pomocí dodávaného skriptu "script.rugged" a výsledek byl uložen v PLA tabulce. Tu jsem potom načetl vytvořeným skriptem a celkem jednoduše, podle definice tabulky, zjistil na kolik vstupů je každý výstup citlivý.

5.2 Test č.1

Tento test je jen jeden z vybraných testů, podle kterých se vylepšovalo nastavení parametrů genetického algoritmu. Zaměřuje se na velikost populace a má za úkol zjistit jak musí být populace velká, aby byl program nejefektivnější. Tento test byl proveden generátorem, ve kterém se ještě do nově vznikající generace nevkládali nově vytvořená triviální řešení.

5.2.1 Nastavení

Rozhodl jsem se velikost populace zkoumat na problému, který bude obsahovat skoro všechny optimalizační kritéria a bude rozumně velký. Velikost populace jsme postupně nastavili na hodnoty 10, 30, 50, 70, 90, 120. Výsledky jsme průměrovali z minimálně 10-ti měření. Podrobné naastavení parametrů generátoru uvádí tabulka 5.1

5.2.2 Naměřené výsledky

V tabulce 5.2 jsou průměrně dosažené hodnoty fitness funkce po pevně zadaném počtu 500ti generací.

Na obrázku 5.1 uvádím průběh fitness v náhodně vybraných výsledcích — vždy jeden pro jednu velikost populace.

Parametr	hodnota
počet vstupů	10
počet výstupů	1
počet hradel	100
testovatelnost	0.1 s váhou 5
řiditelnost	0.7
maximální větvení	4
počet úrovní	25
maximální fan-in	AND,NAND,OR,NOR — 4
kontrola počtu hradel	ANO
konec algoritmu	pevně po 500 generacích

Tabulka 5.1: Test č.1 - nastavení parametrů

Velikost populace	průměrná fitness
10	7.187
30	7.368
50	7.421
70	7.441
90	7.458
120	7.440

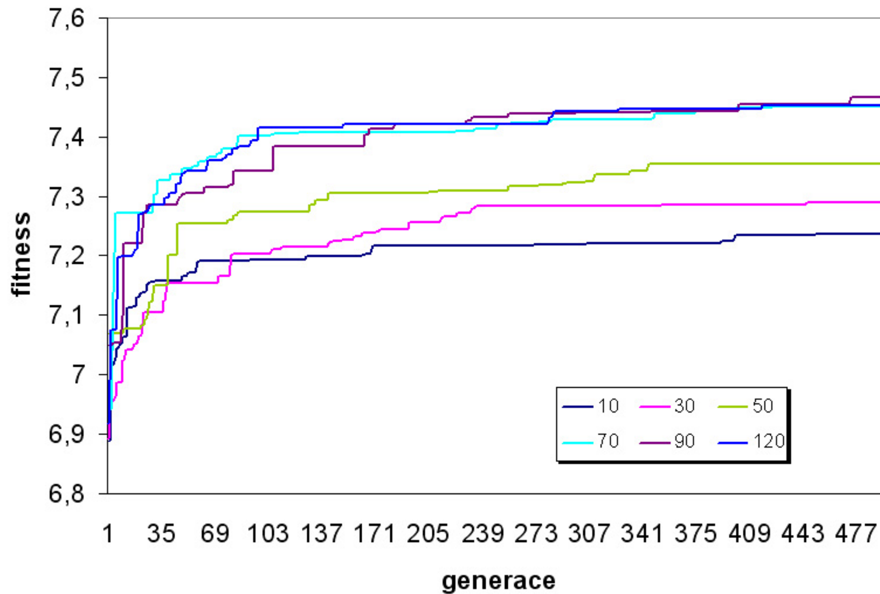
Tabulka 5.2: Test č.1 - výsledky

5.2.3 Vyhodnocení

Jak je vidět, generátor zhruba od velikosti populace 70 generuje srovnatelné výsledky, jak v kvalitě, tak v průběhu konvergence. Tuto hodnotu jsem ještě náhodně otestoval na jiných zadáních a experimentálně ji tím ověřil. Poněvadž jsem později do generátoru přidával funkci, která do nové populace vkládá nové jedince, tak jsem do finálního generátoru zvolil pro velikost populace pevnou hodnotu 80.

5.3 Test č.2

Tento test se snažil otestovat funkci genetického algoritmu co se týká optimalizace obvodu na zadanou testovatelnost.



Obrázek 5.1: Test č.1 - Průběh fitness funkce pro různá nastavení velikosti populace

5.3.1 Nastavení a metodika testu

Generoval jsem 3 druhy obvodů, každý po 50ti vzorcích.

1. druh — Testovatelnost = 0
2. druh — Testovatelnost = 0.1
3. druh — Testovatelnost = 0.2

Ostatní nastavení přepínačů generátoru je uvedeno v tabulce 5.3.

Toto nastavení bylo zvoleno po předběžných testech, aby se snažilo simulovat reálné obvody. Přepínač na kontrolu hradel spolu se zvýšenou váhou testovatelnosti, je kvůli zamezení zvýhodňování dobře testovatelných hradel (XOR,XNOR,NOT) v průběhu optimalizace, na úkor ostatních. Tento nepříjemný efekt popíši v závěru celé práce.

5.3.2 Naměřené výsledky

V tabulce 5.4 sloupeček "test" označuje testovatelnost obvodu, "faults" označuje celkový počet poruch obvodu, "r_faults" vyjadřuje počet redundandtních poruch - nedetekovatelných, "ratio" je podíl $\frac{faults}{r_faults}$ a zhruba vyjadřuje jak moc je obvod redundantní.

-i 20	Počet vstupů — 20
-o 2	Počet výstupů — 2
-n 80	Počet hradel — 80
-m 15 4	Maximální fan-in AND,NAND,OR,NOR — 4
-w 16 16	Váha testovatelnosti zvýšena na 16ti násobek
-cg	Zapnuta kontrola správného počtu hradel
-G 2000	Po 2000 generacích ukonči generování.
-g 500	Nebo pokud za posledních 500 generací se nejsilnější jedinec nezměnil

Tabulka 5.3: Test č.2 - Parametry generátoru

"max ratio" označuje maximální "ratio" ze všech průměrovaných hodnot. Sloupeček "influence" označuje na kolika vstupech je závislý výstup obvodu. Poněvadž byli generovány dvouvýstupové obvody, je tato hodnota počítána jako průměr z obou dvou. Všechny tyto hodnoty jsou zprůměrovány z 50ti vzorků.

test	**faults**	**r_faults**	**ratio**	**max ratio**	**influence**
0.1	293	126	$0,43$	$0,87$	$10,57$
0.1	256	102	$0,40$	$0,80$	$11,43$
0.2	256	87	$0,34$	$0,59$	$12,58$

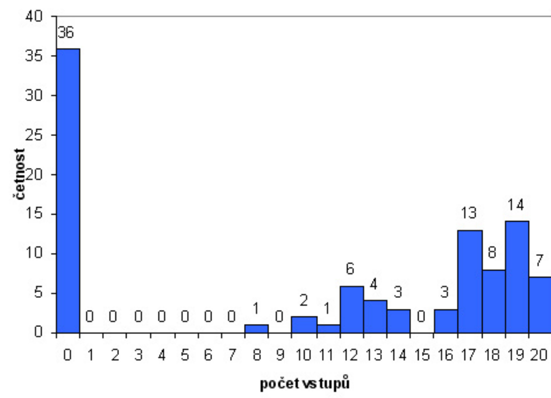
Tabulka 5.4: Test č.2 - Výsledky měření

V grafu 5.2 uvádím nezprůměrované rozložení výskytu hodnoty "influence" pro testovatelnost rovnu 0. Pro zbylé 2 testovatelnosti jsou v grafu 5.3 resp. v grafu 5.4.

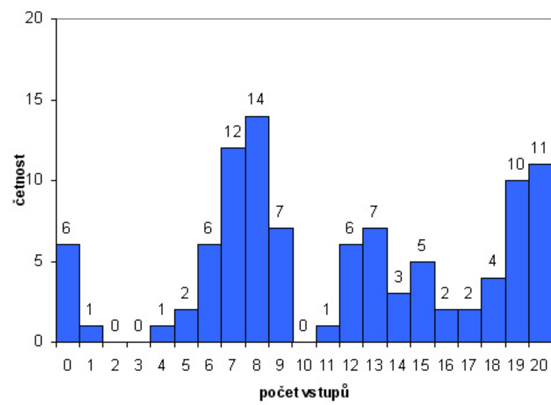
Graf 5.5 ukazuje přibližnou lineární závislost mezi redundancí obvodu, vyjadřenou pomocí poměru počtu poruch v obvodě a počtu nedetekovatelných poruch, a hodnotou, která vyjadřuje závislost výstupu na počtu vstupů. V grafu jsou uvedeny naráz všechny hodnoty z měření. Průměrován je akorát vliv vstupů na více výstupů v rámci jednoho obvodu.

5.3.3 Vyhodnocení

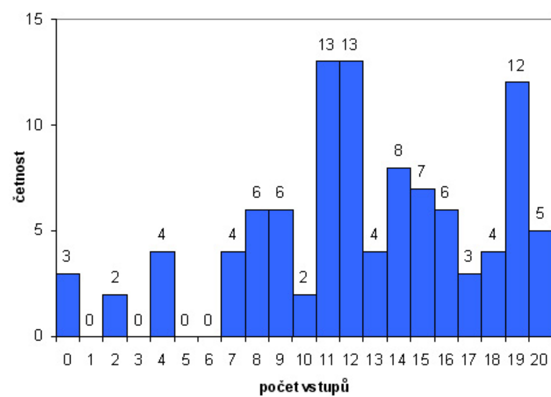
Částečně se potvrdilo, že reálná testovatelnost obvodu a s tím spojená jeho redundance, je ovlivněna volbou vstupního parametru "testovatelnost", avšak nikterak výrazně. Z grafů četností je vidět, že při generování obvodů s nastavenou alespoň nějakou testovatelností,



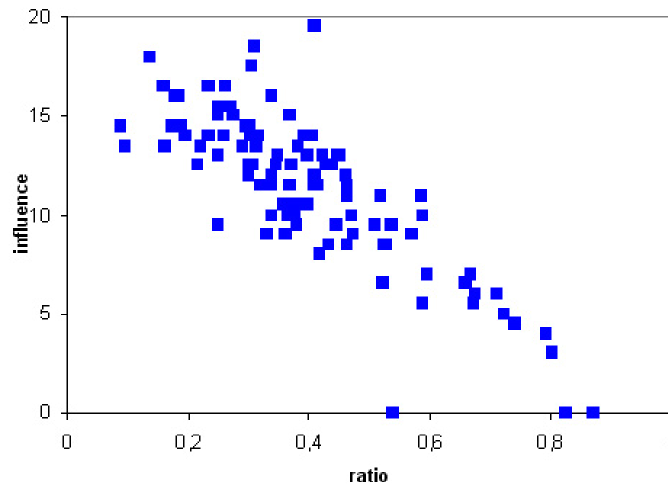
Obrázek 5.2: Test č.2 - Četnost "influence" - testovatelnost 0



Obrázek 5.3: Test č.2 - Četnost "influence" - testovatelnost 0.1



Obrázek 5.4: Test č.2 - Četnost "influence" - testovatelnost 0.2



Obrázek 5.5: Test č.2 - Závislost "influence" na "ratiu" obvodu

dochází k mnohem méně případům, že výstup není citlivý na žádný vstup. Dá se říci, že citlivost obvodu stoupá s testovatelností, avšak rozptyl výsledných hodnot v rámci jednoho nastavení je tak veliký, že se nemůžeme na nastavený vstupní parametr nikterak spolehnout.

V grafu 5.5 se nám ukázala jistá závislost redundance obvodu s citlivostí výstupu, což je logické. Pokud nebude výstup citlivý ani na jednu vstupní proměnou, nebude možné žádným vstupním vektorem otestovat většinu možných poruch v obvodě — budou všechny redundantní.

5.4 Test č.3

Tento test je zaměřen na kontrolu redundancí v obvodě a na možnost resytnézy v závislosti na počtu hradel v obvodu.

5.4.1 Nastavení

Tento test měl nastaveny pevné parametry podle tabulky 5.3. Měnili jsme parametr počet hradel a to v těchto krocích 30, 80, 150, 250, 400 a 600. Tím jsme vlastně odškálovali jakousi "mohutnost" obvodu. Všechny obvody jsme vygenerovali ve 20-ti verzích a dosažené výsledky jsme průměrovali.

Zde musím podotknout, že zvolenou testabilitu dosahovali akorát obvody do 150 hradel.

-i 20	Počet vstupů — 20
-o 1	Počet výstupů — 1
-m 15 4	Maximální fan-in AND,NAND,OR,NOR — 4
-w 16 5	Váha testovatelnosti zvýšena na 5ti násobek
-tt 0.1	Testovatelnost obvodu nastavena na 0.1
-tc 0.7	Řiditelnost obvodu nastavena na 0.7
-cg	Zapnuta kontrola správného počtu hradel
-G 1000	Po 2000 generacích ukonči generování.
-g 400	Nebo pokud za posledních 500 generací se nejsilnější jedinec nezměnil

Tabulka 5.5: Test č.3 - Parametry generátoru

Větší obvody už se hranici 0.1 ani nepřiblížili. To je způsobeno tím, že pro větší obvod je složité pozorovatelnost propagovat přes hradla dále do obvodu. Z toho plynulo, že vlastně obvody nad 150 hradel se generovali s co největší testovatelností. Zajímavé bylo že řiditelnost 0.7 dosáhli úspěšně všechny obvody. Menší obvody ji měli spíše tendenci překračovat.

Zároveň jsme vygenerovali v 5-ti vzorcích jejich 10-ti výstupové varianty. Na nich jsme zkoumali rozdílnou shopnost resytnézy obvodu.

5.4.2 Naměřené výsledky

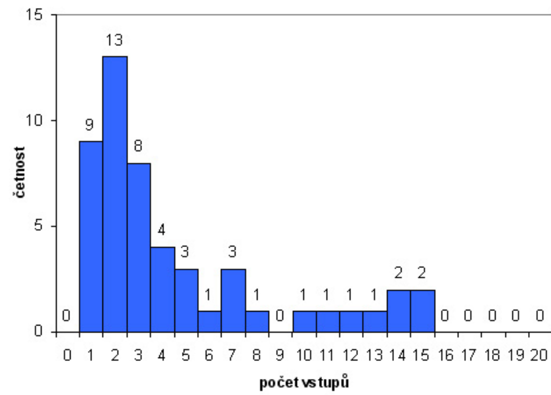
V tabluce 5.6 uvádím výsledky měření vzorků s jedním výstupem. Sloupeček označený "range of ratio" Udává minimalní — maximální "ratio" vyskytující se v testovaných vzorcích. V tabulce 5.7 je rozepsána citlivost nebo-li "influence" výstupů v každém vzorku.

počet hradel	**faults**	**r_faults**	**range of ratio**	**ratio**	**influence**
30	105	23	0,01 — 0,92	0,22	15,15
80	273	120	0,18 — 0,78	0,44	14,90
150	484	204	0,19 — 0,76	0,42	18,75
250	801	464	0,27 — 0,97	0,58	17,85
400	1256	657	0,27 — 0,86	0,52	18,00
600	1861	1022	0,39 — 0,76	0,55	20,00

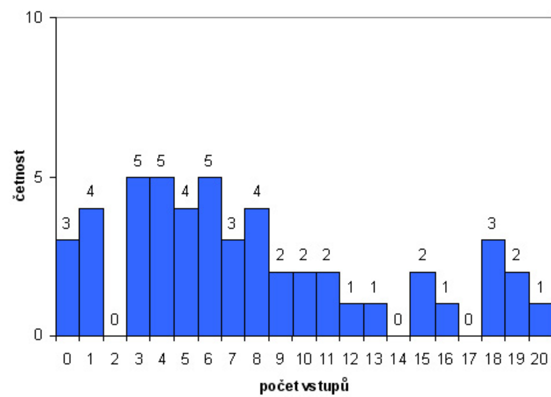
Tabulka 5.6: Test č.3 - Naměřené výsledky - 1 výstup

počet hradel	**1.**	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**	**10.**	**11.**	**12.**	**13.**	**14.**	**15.**	**16.**	**17.**	**18.**	**19.**	**20.**
30	18	20	13	18	18	16	17	19	13	14	11	12	18	17	11	17	17	18	16	0
80	17	16	18	17	20	18	9	10	9	12	8	14	20	20	20	19	7	9	20	15
150	20	20	20	20	20	15	19	18	19	20	17	19	19	20	18	12	20	20	20	19
250	20	18	18	20	18	20	0	20	18	14	20	19	17	17	20	19	20	19	20	20
400	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	0	20	0	20	20
600	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20

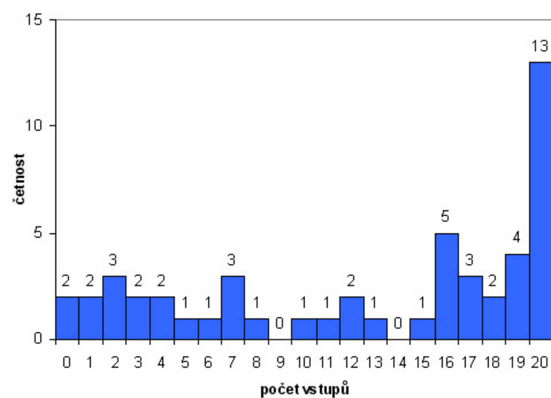
Tabulka 5.7: Test č.3 - Citlivost výstupu v jednotlivých vzorcích - 1 výstup



Obrázek 5.6: Test č.3 - Četnost "influence" - 10 výstupů - 30 hradel



Obrázek 5.7: Test č.3 - Četnost "influence" - 10 výstupů - 150 hradel



Obrázek 5.8: Test č.3 - Četnost "influence" - 10 výstupů - 600 hradel

Obdobně uvádím tabulku 5.8, ve které jsou výsledky 10-ti výstupových obvodů. V grafech 5.6, 5.7 a 5.8 jsou uvedeny četnosti citlivostí všech výstupů ve všech vzorcích pro nastavení 30, 150 a 600 hradel.

počet hradel	**faults**	**r_faults**	**ratio**	**influence**
30	118	18	$0,15$	$4,58$
80	298	60	$0,20$	$6,88$
150	519	137	$0,26$	$7,70$
250	835	265	$0,32$	$10,48$
400	1328	469	$0,35$	$11,56$
600	1919	655	$0,34$	$12,76$

Tabulka 5.8: Test č.3 - Naměřené výsledky - 10 výstupů

5.4.3 Vyhodnocení

Z testu je patrné, že velikost generovaného obvodu úzce souvisí s jeho testovatelností, případně s jeho resyntézou. Jsou zde vidět především obrovské rozptyly redundance genrovaných obvodů a spíše její konvergenci k 50-ti procentům se zvětšujícím se obvodem. Zajimavý byl fakt, že rostla citlvost výstupů v obvodě s jeho velikostí a tím související rostoucí průměrnou redundancí. Je to opačný závěr, než-li v minulém testu a přisuzuji to již zmíněnému velikému rozptylu redundance a průměrování hodnot. Z tabulky **??** je totiž patrné, že se při větších obvodech generovali již ve všech případech, buď obvody s plnou citlivostí a nebo s nulovou. Na závěr uvádím, že pozorovaná citlivost vícevýstupových hradel byla výrazněji citlivá na velikost.

5.5 Test č.4

V tomto testu se zabývám porovnáním již hotového souboru kombinačních obvodů vhodných na testování — ISCAS s obvody podobných parametrů vygenerovaný programem gennetwork. Kombinační obvody v ISCAS jsou reálné obvody. Převážně jde o řadiče, nebo obvody realizující nějakou kontrolu, ojediněle specializovaná sčítačka. Test měl za úkol zhodnotit, jestli lze generátorem generovat obvody, které se nějakým zůsobem podobojí reálným obvodům.

5.5.1 Generování obvodů

První problém byl, že nikde nebyly parametry těchto obvodů blíže specfikovány, tudíž jsme si je museli získat přímo ze souboru popisujícího daný obvod. Především šlo o počet všech typů hradel, kolik mají maximálně vstupů a další. Hradla BUF, které se v ISCAS vyskytují, gennetwork nepoužívá a proto tato hradla vypustil. Naopak v žádném zkoumaném obvodu ze souboru ISCAS nebylo hradlo XNOR, tak bylo nutné nastavit pravěpodobnost výskytu tohoto hradla na 0. Parametry, které jsem zjistil uvádím v tabulce 5.9. Všechny položky znamenají počet výskytů příslušného hradla, pokud se objevuje druhá hodnota v závorce, tak znamená jiný maximální fan-in příslušného hradla nežli standartní 2.

Obvody jsem generoval s optimalizačním kritériem nastaveným na počet, s pravděpodobnostmi a maximálními fan-iny hradel přesně podle tabulky 5.9. Zbytek parametrů generátoru bylo ponecháno standartně.

Vygeneroval jsem zhruba 30 setů logických sítí s podobnými parametry jako kombinační část set ISCAS. Z těch jsem se snažil pomocí testů vybrat co nejvíce podobné obvody. Na závěr pak zhodnotit kvalitu těchto generovaných obvodů jako benchmarkového setu.

5.5.2 Metodika testu

Nově vygenerovanou sadu obvodů jsem porovnával s originálními obvody ISCAS v počtu redundantních poruch a reálné testovatelnosti. Metody jsou stejné jako píši v úvodu testů. Pro reálnou testovatelnost byl každý obvod testován 50× ATPG vždy s náhodným polynomem a seedem.

5.5.3 Naměřené výsledky

Tabulka 5.10 má vždy v lichých řádkách uvedeny původní obvody ze souboru ISCAS. Hned pod ním jsou uvedeny výsledky měření nově vygenerovaného příslušného obvodu. V první části tabulky jsou výsledky atalanty v automatickém módu. V druhé jsou zprůměrované výsledky simulačního módu, kterým byl každý obvod 50 krát testován tisícem náhodných vektorů. Poslední sloupeček označuje podíl nalezených poruch k počtu poruch o kteých víme, že nejsou nedetekovatelné. Vyjadřuje to přibližně, jak moc jednoduše jde daný obvod testovat náhodnými vektory.

název obvodu	**# vstupů**	**# výstupů**	**# hradel**	**# AND**	**# NAND**	**# OR**	**# NOR**	**# XOR**	**# NOT**
*c*17	5	2	7	0	6	1	0	0	0
*c*432	36	7	161	4 (9)	79 (4)	1	19	18	40
*c*499	41	32	203	56 (5)	0	3	0	104	40
*c*880	60	26	358	117 (3)	87 (4)	30	61	0	63
*c*1355	41	32	515	56 (4)	416	3 (4)	0	0	40
*c*1908	33	25	719	63 (8)	377 (8)	1	1	0	277
*c*2670	233	140	998	333 (5)	254	78 (5)	12	0	321
*c*3540	50	22	1447	498 (4)	298 (4)	93 (4)	68 (8)	0	490
*c*5315	178	123	1995	718 (8)	454	215 (5)	27 (3)	0	581
*c*6288	32	32	2417	256	0	1	2128	0	32
*c*7552	207	108	2979	776 (5)	1028	245 (5)	54 (4)	0	876

Tabulka 5.9: Test č.4 - Parametry obvodů ze souboru ISCAS

name	faults	d_faults	r_faults	ratio	nr_faults	RPDF	$\frac{RPRDF}{nr_faults}$
c17	22	22	0	0,00	22	17	0,79
	22	21	1	0,05	21	15	0,71
c432	524	520	1	0,00	523	496	0,95
	494	456	38	0,08	456	384	0,84
c499	758	750	8	0,01	750	722	0,96
	905	891	14	0,02	891	844	0,95
c880	942	942	0	0,00	942	896	0,95
	1128	977	140	0,12	988	710	0,72
c1355	1574	1566	8	0,01	1566	1474	0,94
	1963	1928	29	0,01	1934	1656	0,86
c1908	1879	1870	7	0,00	1872	1204	0,64
	2561	1284	1180	0,46	1381	868	0,63
c2670	2747	2630	86	0,03	2661	2301	0,86
	3074	2929	144	0,05	2930	2295	0,78
c3540	3428	3291	137	0,04	3291	3206	0,97
	4214	1343	2683	0,64	1531	757	0,49
c5315	5350	5291	59	0,01	5291	5046	0,95
	6933	4865	1900	0,27	5033	2670	0,53
c6288	7744	7710	34	0,00	7710	7589	0,98
	7184	7078	22	0,00	7162	6613	0,92
c7552	7550	7418	71	0,01	7479	6716	0,90
	8536	7506	924	0,11	7612	4343	0,57

Tabulka 5.10: Test č.4 - Naměřené výsledky

5.5.4 Vyhodnocení

Celkem 7 z 11-ti generovaných obovodů, mohu prohlásit za podobné obvodům ze souboru ISCAS, podle testovaných parametrů. Nenašel jsem mezi nepodařenými obvody žádnou souvislot, která by indikovala, proč se je nepodařilo replikovat. Poněvadž jsou obvody z ISCAS v reálu používané a byly navrženy na nějakou konkrétní funkci, je velkým úspěchem, že jsme generátorem náhodných obvodů vygenerovali obvody (v rámci testovatelnsoti) podobné.

6 Závěr

Cílem této práce bylo navrhnout generátor, který bude umět generovat náhodné logické obvody popsané Booleovskou funkcí, stavovým automatem a sítí hradel. Měl být kladen důraz na co největší náhodnost generovaných obvodů v rámci množství volitelných parametrů, z nichž některé bylo možné počítat až z vygenerovaného obvodu. Proto byly při generování použity některé pokročilé iterativní metody.

Návrh i implementace všech částí generátoru se úspěšně zdařila a program pracuje podle požadavků. Testování bylo zaměřeno na vliv klíčových volitelných parametrů na celkovou testovatelnost generovaných obvodů. Zde se projevili slabé stránky generátoru. Vliv zadané testovatelnosti má nepatrný efekt na skutečnou testovatelnost obvodu. S tím úzce souvisí generování velkých redundantních částí obvodu, které se objevuje náhodně, s pouze nepatrným vlivem vstupních parametrů. Rozptyl v citlivosti (testovatelnosti) generovaných obvodů je tak velký, že se nemůžeme spolehnout na volbu vstupních parametrů.

Toto si vysvětluji špatnou volbou metody, která se stará o výpočet testovatelnosti. Jelikož jí počítá z plošných koeficientů hradel a z jejich bezprostředních okolí, tak nemůže zohlednit redundantní zapojení částí obvodu a tak ovlivnit jejich tvorbu. Dále je to pak ovlivněno čistě náhodným generováním inicializačních obvodů bez dodatečných sofistikovaných kontrol redundancí. Poněvadž je velmi malá pravděpodobnost že se trefíme do rozumného obvodu bez redundancí, který by pak mohl křížením tvořit kvalitnější jedince.

Dalším objeveným problémem při generovaní sítí hradel bylo, že při zadané optimalizaci podle testovatelnosti se generuje nadměrné množství jednoduše testovatelných hradel (XOR,XNOR,NOT). Proto doporučuji používat zároveň kontrolu počtu hradel podle generované pravděpodobnosti. To omezí problém s lehce testovatelnými hradly pouze na jejich přesun v obvodě k začátku/konci.

6.1 Náměty na vylepšení

Zde uvádím některé náměty na vylepšení, které by měly odstraňovat popsané slabé stránky generátoru.

- doimplementovat jiné metody výpočtu testovatelnosti
- zjišťovat v průběhu genetiky reálnou testovatelnost i pomocí ATPG
- kontrolovat různým způsobem redundanci/citlivost obvodu.

7 Literatura

[1] Atalanta-M webpage.
http://service.felk.cvut.cz/vlsi/prj/Atalanta-M/.

[2] Berkeley Logic Interchange Format (BLIF). University of California, Berkeley
http://service.felk.cvut.cz/vlsi/prj/Circ_Gen/blif.pdf.

[3] download page at VLSI ČVUT.
http://vlsi.felk.cvut.cz/index.php?page=download.

[4] Problémy a algoritmy - webová stránka předmětu.
http://service.felk.cvut.cz/courses/36PAA/.

[5] SIS 1.2 homepage.
http://embedded.eecs.berkeley.edu/Alumni/pchong/sis.html.

[6] Wikipedia - Genetic algorithm.
http://en.wikipedia.org/wiki/Genetic_algorithm/.

[7] G. R. C. M. Maunder, R.G. Bennetts. CAMELOT: A Computer-Aided Measure for Logic Testability. In *Proceedings of Intenational Conference on Computer Communication*, pages 1162–1165, 1980.

[8] L. H. Goldstein and E. L. Thigpen. SCOAP: SANDIA CONTROLLABILITY/OBBERVABILITY ANALYSIS PROGRAM*. 1979.

[9] J. Hlavička. *Diagnostika a spolehlivost*. Vydavatelství ČVUT, 2th edition, 1998.

[10] T. Měchura. *Parametrizovaný generátor náhodných booleovských funkcí*. 2006.

[11] J. Strnadel. Normalized testability measures based on rtl digital circuit graph model analysis. In *Proceedings of The fifth International Scientific Conference Electronic Computers and Informatics 2002*, Edition 55, pages 200–205. The University of Technology Košice, 2002.

[12] J. Strnadel. *Analýza a zlepšení testovatelnosti číslicového obvodu na úrovni meziregistrových přenosů*. PhD thesis, 2004.

[13] J. D. V. Jáneš. *Logické systémy*. Vydavatelství ČVUT, Praha, 2001.

A Uživatelská příručka

A.1 genpla

Syntaxe:

`genpla -i n -o n -t n [options] [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%.
-odt 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.
-ND	Neprovádí se kontrola proti duplicitním termům.
-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 genpla

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:

```
genpla -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

```
genpla -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.

A.2 genstate

Syntaxe:

`genstate -i n -o n -t n -s n [options] [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.
-s n	Nastav počet generovaných stavů na n.
-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%.
-odt n	Nastav poměr 1:0 ve výstupní matici na n %. Standartní hodnota je 50%.
-sr	Přidej do obvodu jeden vstup, kterým se resetuje automat Zároveň přidá jeden přechod, který se počítá do hodnoty zadané přepínačem -t. Ten označuje přechod, kterým se automat vyresetuje.
-se n	Nastav počet tříd ekvivalence na n.
-sp n	Nastav n % přechodů automatu na neurčitý. Vytvoří tzv. pseudoekvivalentní stav.
-timeout n	Nastav timeout na n sekund. Standartní hodnota je 1000 s.
-NC	Neprovádí se kontrola konzistence termů.
-ND	Neprovádí se kontrola proti duplicitním termům.
-NS	Neprovádí se kontrola spojitosti grafu automatu. V automatu můžou vzniknout nedostupné části.
-NK	Neprovádí se kontrola koncovosti každého stavu. Může se vygenerovat automat se stavem ze kterého nepovedou žádné přechody.
-NZ	Neprovádí se kontrola dostupnosti každého stavu. Může nastat že do některého stavu nepovede žádný přechod.
-ND	Neprovádí se kontrola determinismu. Může se nastat situace, že pro stejné ohodnocení vstupů bude mít automat nastaveny přechody do různých stavů.
file_name	výstupní soubor. Pokud není specifikován, tak se automat ve formátu KISS vypíše na standartní výstup.

Tabulka A.2: Popis přepínačů programu genstate

Program generuje náhodný stavový automat ve formátu KISS podle zadaných parametrů. Generátor se může při jistých nastaveních dostat do smyčky, kdy již nedokáže kvůli kontrolám vygenerovat další přechod. Tento stav není schopen kvůli povaze systému odhalit. Proto je v programu implementován timeout. Při jeho vypršení se program ukončí. Standartně generuje deterministický automat. Pokud mu zadáme více termů než má odpovídající deterministický úplný automat, tak počet termů omezí.

Příklady:

```
genstate -i 10 -o 2 -s 8 -t 100 out.kiss
```

Generuje deterministický stavový sutomat s 10ti vstupy, 2ma výstupy, 8mi stavy. Automat bude mít 100 přechodů a uloží se do souboru out.kiss

```
genstate -i 10 -o 2 -s 5 -t 20 -idc 20 -ND -NC -sr
```

Generuje nedeterministický automat. Bude mít 10 vstupů + 1 resetující vstup, 2 výstupy, 5 stavů, 19 přechodů a jeden přechod ze všech stavů do stavu RESET, každý term bude mít na 20ti procentech vstupech don't care. Automat se vypíše na standratní výstup.

```
genstate -i 10 -o 2 -s 10 -t 80 -se 7
```

Vygeneruje deterministický stavový automat, který vypíše na standartní výstup. Automat bude mít 10 vstupů, 2 výstupy, 10 stavů, 80 termů a 7 tříd ekvivalence. Tzn. že 3 stavy budou redundantní.

A.3 gennetwork

Syntaxe:

`gennetwork -i n -o n -n n [options] [file_name]`

Ve všech parametrech se za n, g a x vkládá celé kladné číslo, místo d se použije realné číslo. n znamená nějaký počet, místo g se zadává číslo označující výběr typu hradla, x má speciální význam — určený u parametru.

Typy hradel — hradla mají tyto hodnoty: AND-1, NAND-2, OR-4, NOR-8, XOR-16, XNOR-32, NOT-64. Více typů hradel se volí jako součet jejich hodnot. např.: Všechny hradla mimo XOR a XNOR: 1001111b=79. Takže za g dosadíme číslo 79.

Povinné parametry jsou uvedeny v tabulce A.3.

-i n	Nastav počet vstupů obvodu na n.
-o n	Nastav počet výstupů obvodu na n.
-t n	Nastav počet všech hradel v obvodu na n.

Tabulka A.3: Popis přepínačů programu gennetwork - část 1.

Volitelné parametry jsou uvedeny v tabulce A.4.

Berte v úvahu, že některé kombinace přepínačů mohou vést k situaci, kdy nebude možné vygenerovat obvod. Jedná se především o kombinaci hodně výstupů s málo hradly, nebo hodně hradel spolu s malou hodnotou maximálního větvení. Program také nekontroluje nesmyslné zadávání parametrů, jako například záporné hodnoty apod.

Pravděpodobnost jednotlivých hradel je počítána z jejich indexů. A to takto: Pravděpodobnost hradla typu i je rovný podílu jeho indexu a součtu indexů všech typů. Standartně jsou indexy nastaveny na 5 u všech typů hradel. Takže abychom generovali hradlo AND v 50ti procentech. Tak musíme jeho index nastavit na 30. $0.5 = \frac{30}{30+5+5+5+5+5+5}$.

-l n	Nastav úroveň obvodu na n.
-c n	Nastav stupeň maximálního větvení na n. Používat opatrně! Nízké hodnoty v kombinaci s jinými parametry mohou vést k neschopnosti generovat jakýkoliv obvod. Program se může zaseknout nebo se ukončit vyjímkou. Standartně je nastaveno 255.
-m g n	Nastav maximální fan-in hradla g na hodnotu n Standartně je nastaven na 2 pro všechny hradla.
-p g n	Nastav index pravděpodobnosti hradla g na hodnotu n. Standartně je nastavena hodnota 5. Pokud se nastaví 0 tak se hradlo g nebude generovat.
-s x	Nastav výstupní formát na x. Může se použít kombinace hodnot x= 1-BENCH, 2-BLIF, 4-SLIF. Standartně je nastaven výstup BENCH a BLIF. Každý formát si přidá k file_name svojí koncovku.
-tc d	Nastav požadovanou řiditelnost na $d < 0; 1 >$
-to d	Nastav požadovanou pozorovatelnost na $d < 0; 1 >$
-tt d	Nastav požadovanou testovatelnost na $d < 0; 1 >$
-w x d	Nastav váhu složky x na d ve fitness funkci. Složky: 1-úrovně,2-# hradel,4-řiditelnost,8-pozorovatelnost,16-testovatelnost Standartně je váha všech složek nastavena na 1.
-g n	Nastav počet dynamických generací na n. Standartně je 50. Program končí, když po dobu n generací se nejsilnější jedinec nezmění.
-G n	Nastav počet statických generací na n. Standartně není nastaveno. Program se ukončí po projití n generací, funguje společně s -g
-P n	Nastav velikost populace genetického algoritmu. Standartně je 80.
-cg	Zapne optimalizační kritérium, které kontroluje počet hradel v obvodě. Realný stav srovnává s předpočítaným stavem z indexu pravděpodobností
-info	Zapne výpis informací o generovaném obvodě na konci programu do chybového výstupu. Zároveň vypisuje info do výstupních souborů.
file_name	Zvol jméno souboru pro výstup obvod ve formátech zvolených přepínačem -s. Pokud se nezadá tak se vše píše do standartního výstupu.

Tabulka A.4: Popis přepínačů programu gennetwork - část 2.

Příklady:

```
gennetwork -i 10 -o 2 -n 20 out
```

Vygeneruje triviálně zapojený obvod, který má 10 vstupů, 2 výstupy a bude tvořen 20ti maximálně 2vstupovými hradly. Hradla budou generována se stejnou pravděpodobností. Výstup uloží ve formátu BENCH do souboru out.bench a ve formátu BLIF do souboru out.blif.

```
gennetwork -i 10 -o 2 -n 30 -m 3 4 -l 5 -s 1 -g 200
```

Vygeneruje obvod s 10ti vstupy, 2ma výstupy s 30 ti hradly, ve kterých bude hradlo AND a NAND maximálně 4vstupvé, ostatní hradla budou maximálně 2vstupová. Všechny hradla budou generovány se stejnou pravděpodobností. Genetický algortimus bude obvod šlechtit, aby měl 5 úrovní hradel. Výstup je ve formátu BENCH do standartního výstupu. A optimalizace se ukončí, pokud 200 generací se nezmění nejsilnější jedinec v populaci.

```
gennetwork -i 10 -o 2 -n 50 -p 48 0 -tc 0.5 -w 4 10 -cg -G 1000 out
```

Vygeneruje obvod s 10ti vstupy, 2ma výstupy s 40 ti hradly. Bude generovat všechna hradla se stejnou pravděpodobností mimo XOR a XNOR, které se nebudou generovat vůbec. Optimalizovat se bude podle řiditelnosti nastavenou na 0.5 bude mít 10x větší vliv, než optimalizace podle počtu hradel. Konec algortimu bude po provedení 1000 generací nebo pokud se nezmění nejsilnější jedinec po dobu 50ti generací.

Poznámka

Při zadávání optimalizace podle jedné z testovatelnosti, je vhodné použít i optimalizační kritérium -cg a nastavit mu zhruba 20× menší váhu. Bez kontroly hradel, se totiž po nějaké době stává, že začnou převažovat hradla, která mají lepší testovatelnost (XOR,XNOR,NOT). Tyto hradla se většinou pak přesouvají k začátku nebo konci obvodu, aby řídítelnost nebo pozorovatelnost propagovali hlouběji do obvodu.

B Obsah přiloženého CD

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

src - zde jsou uloženy zdrojové soubory generátorů

texty - složka obsahuje tuto práci v elektronické podobě

testy - obsahuje veškeré materiály týkající se provedených testů

[1] [12] [11] [9] [4] [6] [1] [8] [13]