Wydział Podstawowych Problemów Techniki
Politechnika Wrocławska

Translator i symulator kwantowych sieci logicznych

Katarzyna Marek
Nr indeksu: 236480

Praca inżynierska napisana
pod kierunkiem
dr Macieja Gębali

Politechnika
Wrocławska

Wrocław 2019

Spis treści

Wstęp

W latach 70. po raz pierwszy zostało użyte sformułowanie ”kwantowej teorii informacjiśuregujące użycie efektów kwantowych do manipulacji informacją. Niedługo potem pojawiła się idea układów kwantowych, które analogicznie do układów logicznych mają pozwalać na przeprawadzanie obliczeń. W 1994 temat komputerów kwantowych stał się bardzo interesujący, gdy Peter Shor opublikował algorytm wykorzystujący układy kwantowe, który rozwiązuje problem faktoryzaji w czasie wielomianowym. Najlepsze znane algorytmy na komputery klasyczne wymagają czasu wykładniczego. Pomimo, że fizyczne budowanie układów kwantowych rozwija się powoli to podstawy teoretyczne są są od dawna dobrze rozwinięte.

Układy kwantowe znacznie różnią się od klasycznych układów logicznych. Chociaż analogicznie do klasycznych bitów operują na kubitach oraz składają się z bramek kwantowych tak jak układy logiczne z bramek logicznych to bramki kwantowe różnią się od bramek klasycznych, a kubity mogą nieść znacznie więcej informacji niż klasyczne bity. Pomimo różnic każdy problem rozwiązywany przez komputer klasyczny może zostać rozwiązany przez komputer kwantowy.

Ta praca zajmuje się zależnościami między komputerami klasycznymi, a kwantowymi. Problemami rozważanymi w tej pracy jest translacja układów logicznych, będących bazą działania komputerów klasycznych, do układów kwantowych oraz symulacja działania układu kwantowego za pomocą komputera klasycznego dla wybranego zestawu bramek kwantowych.

Rozdział 2 zawiera opis logiki klasycznej.

Rozdział 3 jest wprowadzeniem w obliczenia kwantowe i przybliża najważniejsze z nimi związane pojęcia. Następnie prezentuje też zestaw bramek kwnatowych, które będą dalej używane w tej pracy.

Rozdział 4 zawiera definuje cele tej pracy.

Rozdział 5 zawiera schemat architektury systemu oraz opisy poszczególnych komponentów.

Do rozdziału 6 został wydzielony opis zagadanienia translacji.

W rozdziale 7 są omówione szczegóły implementacyjne. Opisane są użyte technologie oraz sposób użycia.

Ostatni rozdział 8 zawiera przykładowe programy wejściowe oraz ich omówienie.

Logika klasyczna

2.1 Funkcja boolowska

Definicja 2.1 *Przez funkcję boolowską, w tej pracy, rozumiemy zupełną funkcję boolowską. Jest to funkcja*

$$f : \{0,1\}^n \to \{0,1\}$$

gdzie $n \in \mathbb{N}$. Wartość 1 jest utożsamiana z logiczną prawdą, a 0 z logicznym fałszem.

Funkcje boolowska możemy zapisać w postaci tabeli prawdy. Na przykład

x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

wtedy $f(0,1,0) = 1$, a $f(0,0,0) = 0$.

2.1.1 Podstawowe funkcje boolowskie

Poniższy zestaw funkcji boolowskich tworzy układ funkcjonalnie pełny.

Definicja 2.2 *Zestaw funkcji boolowskich tworzy układ funkcjonalnie pełny, gdy przy ich użyciu można wyrazić każdą funkcję boolowską.*

Id (identyczność)

$$f(a) = a$$

x	$f(x) = x$
0	0
1	1

Not (negacja)

$$f(a) = \neg a$$

$$f(a) = \overline{a}$$

x	$f(x) = \overline{x}$
0	1
1	0

And (koniunkcja)

$$f(a,b) = a \wedge b$$

$$f(a,b) = ab$$

x	y	$f(x,y) = xy$
0	0	0
0	1	0
1	0	0
1	1	1

Or (alternatywa)

$$f(a,b) = a \vee b$$
$$f(a,b) = a + b$$

x	y	$f(x,y) = x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Xor (alternatywa wykluczająca)

$$f(a,b) = a \otimes b$$

x	y	$f(x,y) = x \otimes y$
0	0	0
0	1	1
1	0	1
1	1	0

2.1.2 Kanoniczna postać sumacyjna (SOP)

Definicja 2.3 *Literał definiuje się jako:*

$$x^e = \begin{cases} x & gdy\ e = 1 \\ \overline{x} & gdy\ e = 0 \end{cases}$$

gdzie $e \in \{0,1\}$, a x symbolem zmiennej.

Definicja 2.4 *Term to produkt literałów (równoważny logicznej koniunkcji), taki, że każda zmienna występuje w nim maksymalnie raz.*

Definicja 2.5 *Term, w którym każda zmienna występuje dokładnie raz, nazywamy mintermem.*

Definicja 2.6 *Kanoniczną postacią sumacyjną (SOP) nazywamy postać funkcji, w której zapisana jest ona jako suma termów (równoważna logicznej alternatywie).*

Każdą funkcję boolowską można zapisać w postaci sumy mintermów w następujący sposób

$$f(a_0, a_1, \ldots a_n) = \sum_{i=0}^{n} b_i * m_i$$

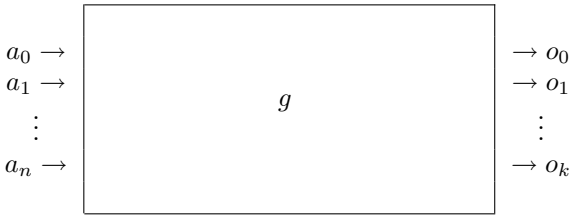
gdzie $b_i \in \{0,1\}$ jest wskaźnikiem (mówi czy dany minterm należy do funkcji f), a m_i to minterm, taki że

$$m_i = a_0^{j_0} a_1^{j_1} \ldots a_n^{j_n}$$

gdzie ciąg $j_0, j_1, \ldots j_n$ to cyfry liczby i zapisanej w postaci binarnej.

2.2 Układy logiczne

Układ logiczny U_g pozwala na obliczanie funkcji $g : \{0,1\}^n \to \{0,1\}^k$, gdzie $n,k \in \mathbb{N}$. Zatem układ ten można zamodelować następująco:



gdzie ciągi $a_0, a_1, \ldots, a_n$ to bity wejściowe, a $b_0, b_1 \ldots, b_n$ to bity wyjściowe, że $\forall i\ a_i, b_i \in \{0, 1\}$ oraz

$$g(a_0, a_1, \ldots, a_n) = (f_0(a_0, a_1, \ldots, a_n), f_1(a_0, a_1, \ldots, a_n), \ldots f_k(a_0, a_1, \ldots, a_n)) = (o_0, o_1, \ldots, o_k)$$

gdzie $f_0, f_1, \ldots, f_k$ to funkcje boolowskie.

Model układu kwantowego

3.1 Stan kwantowy kubitu

Podstawowym nośnikiem informacji w obliczeniach kwantowych, analogicznym do klasycznego bitu, jest bit kwantowy nazywany kubitem. Stan $|0\rangle$ odpowiada klasycznemu 0, a $|1\rangle$ klasycznej 1. Stany $|0\rangle$ i $|1\rangle$ to stany czyste. Kubit może znajdować się w superpozycji tych stanów, w stanie mieszanym.

Mierząc stan kubitu można odkryć jedynie, że jest on $|0\rangle$ lub $|1\rangle$. Stan kubitu po jego zmierzeniu, jeśli był mieszany, ulegnie zmianie i będzie zgodny ze stanem zmierzonym.

Stan kubitu można zapisać jako

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$$

gdzie $\alpha_0, \alpha_1 \in \mathbb{C}$ oraz $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Zapis ten mówi o prawdobodobieństwie otrzymania każdego z możliwych wyników przy pomiarze. Prawdopodobieństwo otrzymania wyniku $|0\rangle$ wynosi $|\alpha_0|^2$, a $|1\rangle$ wynosi $|\alpha_1|^2$.

3.1.1 Notacja Diraca

Zapis '$|\rangle$' nazywany jest notacją Diraca lub bra-ket. Przez $|\psi\rangle$ (nazywanym ket) rozumiemy pewien wektor kolumnowy w przestrzeni Hilberta, gdzie przestrzeń Hilberta jest przestrzenią wektorową nad ciałem liczb zespolonych ze zdefiniowanym iloczynem skalarnym. Wektory

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

,

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

tworzą bazę ortonormalną w 2-wymiarowej przestrzeni Hilberta.

Zatem stan kubitu odpowiada znormalizowanymu wektorowi w 2-wymiarowej przestrzeni Hilberta.

3.2 Stan kwantowy rejestru kubitów

Stan n-kubitowy rejestru kwantowego to

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle + \ldots + \alpha_{2^n-1} |2^n - 1\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}$$

Gdzie $(\forall i \in \{0, 1, \ldots, 2^n - 1\})(\alpha_i \in \mathbb{C})$ oraz

$$\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$$

3.2.1 Stan rejestu kubitów a stanów kubitów

Mając rejestr kwantowy $|\psi\rangle$, którego składowe stany kubitów to $|q_0\rangle$ i $|q_1\rangle$, gdzie

$$|q_0\rangle = \alpha_{00}|0\rangle + \alpha_{01}|1\rangle$$

$$|q_1\rangle = \alpha_{10}|0\rangle + \alpha_{11}|1\rangle$$

Możemy zapisać stan rejestru jako

$$|\psi\rangle = |q_0\rangle \otimes |q_1\rangle = \alpha_{00}\alpha_{10}|00\rangle + \alpha_{00}\alpha_{11}|01\rangle + \alpha_{01}\alpha_{10}|10\rangle + \alpha_{01}\alpha_{11}|11\rangle$$

gdzie przez $\otimes$ rozumiemy produkt tensorowy.

Analogicznie dla rejestru n-kubitowego $|\psi\rangle$, którego stany kubitów to $|q_0\rangle, |q_1\rangle, \ldots |q_n\rangle$

$$|\psi\rangle = |q_0\rangle \otimes |q_1\rangle \otimes \ldots \otimes |q_n\rangle$$

3.2.2 Splątanie kwantowe

N-kubitowy rejestr kwantowy może być w stanie $|\psi\rangle$ takim, że nie istnieją takie stany $|q_0\rangle, |q_1\rangle, \ldots |q_n\rangle$, że

$$|\psi\rangle = |q_0\rangle \otimes |q_1\rangle \otimes \ldots \otimes |q_n\rangle$$

Zjawisko to nazywane jest splątaniem kwantowym.

Przykładem takiego stanu jest stan Bella

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

3.3 Bramka kwantowa

Przez bramkę kwantową rozumiemy operację (o dodatkowych ograniczeniach) wykonywaną na rejestrze kubitów, której wynikiem jest nowy stan rejestru kubitów. Ich działanie jest analogiczne do bramek w układach logicznych.

Każda poprawna bramka kwantowa może zostać zapisana jako macierz unitarna M, to znaczy taka, że $M^\dagger M = I$ oraz $MM^\dagger = I$, gdzie przez $\dagger$ rozumiemy sprzężenie hermitowskie (złożenie operacji transpozycji i sprzężenia zespolonego). N-kubitowa bramka kwantowa M_n jest zatem macierzą unitarną o wymiarach $2^n \times 2^n$ i co więcej każda taka macierz jest poprawną bramką kwantową.

N-kubitowa bramka kwantowa M operuje na n-kubitowym rejestrze kwantowym $|\psi\rangle$ w następujący sposób:

$$\begin{aligned} M|\psi\rangle = M|a_0, a_1, \ldots a_{2^n-1}\rangle &= M(a_0|0\rangle + a_0|1\rangle + \ldots + a_{2^n-1}|2^n-1\rangle) \\ &= a_0 * M|0\rangle + a_0 * M|1\rangle + \ldots + a_{2^n-1} * M|2^n-1\rangle \end{aligned}$$

3.4 Wybrane bramki kwantowe

3.4.1 Bramka NOT

Bramka NOT dokonuje mapowania

$$|0\rangle \rightarrow |1\rangle$$

$$|1\rangle \rightarrow |0\rangle$$

analogicznie do logicznej bramki Not.

Macierzą odpowiadającą temu mapowaniu jest

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Macierz ta jest unitarna.
Bramka Not operuje na kubicie $|\psi\rangle$ w natępujący sposób

$$X|\psi\rangle = X(\alpha_0|0\rangle + \alpha_1|1\rangle) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} \alpha_1 \\ \alpha_0 \end{bmatrix}$$

Bramkę NOT zapisuje się na układzie kwantowym w następujący sposób

3.4.2 Uogólniona bramka Toffoliego

Bramka Feymana

Bramka Faymana inaczej $CNOT$ (z ang. Controlled NOT) jest bramką operującą na 2 kubitach. Dokonuje następującego mapowania

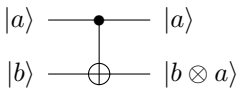
$$|00\rangle \rightarrow |00\rangle$$
$$|01\rangle \rightarrow |01\rangle$$
$$|10\rangle \rightarrow |11\rangle$$
$$|11\rangle \rightarrow |10\rangle$$

Inaczej jej działanie można zapisać w postaci

$$CNOT|a,b\rangle = |a,b \oplus a\rangle$$

Wtedy można patrzeć na nią jak na uogólnioną bramkę XOR.
Bramce tej odpowiadają następująca macierz i zapis w układzie kwantowym:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



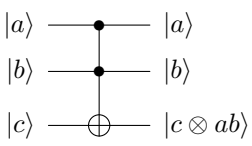
Bramka Toffoliego

Bramką analogiczną do bramki Feymana jest bramka Toffoliego, inaczej $CCNOT$. Nakłada NOT na ostatni kubit (bit wejściowy), jeśli dwa pozostałe (bity sterujące) są w stanie $|11\rangle$. Jej działanie można zapisać następująco:

$$CCNOT|a,b,c\rangle = |a,b,c \oplus ab\rangle$$

Macierz oraz zapis na układzie kwantowym dla tej bramki to:

$$CCNOT = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Uogólniona forma

Bramkę Toffoliego można uogólnić do bramki z n-bitami kontolującymi C_nNOT.

$$C_nNOT\,|a_0, a_1, \ldots, a_n\rangle = C_nNOT\,|a_0, a_1, \ldots, a_n \otimes a_0 a_1 \ldots a_{n-1}\rangle$$

Wtedy dla $n = 2$ mamy bramkę Toffoliego, dla $n = 1$ bramkę Feymana, a dla $n = 0$ bramkę NOT.

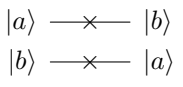
3.4.3 Bramka SWAP

Bramka $SWAP$ operuje na dwóch kubitach zamieniając ich stany.

$$SWAP\,|a,b\rangle = |b,a\rangle$$

Macierz oraz zapis na układzie kwantowym dla tej bramki to:

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



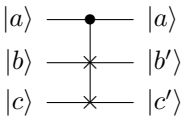
3.4.4 Bramka Fredkina

3-kubitowa bramka Fredkina inaczej $CSWAP$ dokonuje permutacji ostatnich dwóch bitów, wtedy i tylko wtedy gdy bit sterujący jest w stanie $|1\rangle$. Jej działanie można zapisać w następujący sposób:

$$CSWAP\,|c,s_1,s_2\rangle = |c, (\neg c)s_1 + cs_2, (\neg c)s_2 + cs_1\rangle$$

Macierz oraz zapis układzie kwantowym dla tej bramki to:

$$CSWAP = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$



3.4.5 Bramka Hadamarda

Wszystkie wyżej opisane bramki można zaimplementować na klasycznych bitach. Najbardziej podstawowa bramka, która wpowadza kubit w stan superpozycji, to bramka Hadamarda. Dokonuje ona następującego mapowania:

$$H\,|0\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle = |+\rangle$$

$$H\,|1\rangle = \frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle = |-\rangle$$

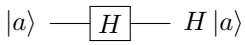
Zarówno w stanie $|+\rangle$ jak i w stanie $|-\rangle$

$$P(|0\rangle) = P(|1\rangle) = \frac{1}{2}$$

gdzie przez $P(|\psi\rangle)$ rozumiemy prawdopodobieństwo otrzymania $|\psi\rangle$ w wyniku pomiaru.

Bramkę Hadamarda można zapisać jako następującą macierz

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$



3.5 Układ kwantowy

Definicja 3.1 *Przez układ kwantowy rozumiemy ciąg bramek kwantowych, które są zdefiniowane jako operacje na n-bitowym rejetrze z nałożonym porządkiem ich wykonywania. Każda z bramek kwantowych wykonywana jest na podzbiorze kubitów z rejestru. Każdy układ kwantowy można wyrazić jako pojedynczą operację, która będącą złożeniem operacji składowych. Operacja ta, jako złożenie macierzy unitarnych, również będzie macierzą unitarną.*

Analiza problemu

4.1 Założenia funkcjonalne

Celem tej pracy jest stworzenie programu, który przyjmuje opis obliczeń (używając predefiniowanego języka) dla układu kwantowego. Natępnie tworzy układ kwantowy odpowidaj tym obliczeniom i symuluje jego działanie.

4.1.1 Założenia dotyczące opisu wejściowego

Opis wejściowy (język wejściowy) powinien pozwalać na niżej opisane działania.

1. Zefiniowanie zmiennej oraz nadanie jej wartości początkowej (0 lub 1).
2. Zdefiniowanie dowolnej funkcji boolowskiej na wcześniej zdefiniowanych zmiennych.
3. Wprowadzenie zmiennej w stan superpozycji, taki, że prawdopodobieństwo otrzymania 0 w wyniku mierzenia wynosi $\frac{1}{2}$.

4.1.2 Założenia dotyczące zwracanego wyniku

Program powinien zwrócić

1. Układ kwantowy w postaci stanu początkowego rejestru oraz serii bramek.
2. Wynik symulacji.

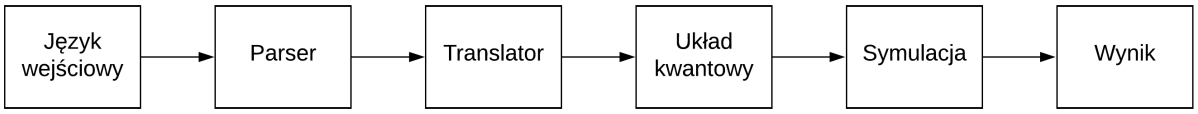
4.2 Założenia niefunkcjonalne

Program powinien starać się optymalizować liczbę używanych dodatkowych kubitów, wymaganych przy przełożeniu funkcji nieodwracalnych na bramki kwantowe.

Projekt systemu

5.1 Architektura Systemu

Na rysunku 5.1 widać diagram potoku systemu. Program na wejściu przyjmuje opis obliczeń wyspecyfikowany przy użyciu z góry zdefiniowanego języka wejściowego. Opis ten jest parsowany na obiekty rozumiane przez język programowania, przy okazji jest sprawdzana poprawność leksykalna oraz semantyczna zdefiniowanych w opisie wejściowym instrukcji. Wszystkie instrukcje są tłumaczone na akcje związane z układem kwantowym, czyli powiększanie wielkość rejestru kubitów (deklaracja) lub operowanie na rejestrze za pomocą bramek kwantowych. Następnie symulowane jest działanie tak powstałego układu kwantowego. Program zwraca schemat utworzonego układu kwantowego oraz wynik symulacji.



Rysunek 5.1: Diagram potoku

5.2 Język wejściowy i Parser

5.2.1 Gramatyka

P	→	**E P** | **E**
E	→	var = **V** | **Q**
V	→	var | const | **K**
W	→	**V** , **W** | **V**
K	→	and(**W**) | not(**V**) | or(**W**) | xor(**W**)
Q	→	hdm(var) | swp(var , var) | tfl(**L** var **R**) | **F**
F	→	frd(: var , var , var) | frd(var , : var , var) | frd(var , var , : var)
L	→	**L** : var , | ϵ
R	→	, : var **R** | ϵ

5.2.2 Analiza leksykalna

Token	Regex	Opis
`var`	`[a-zA-Z_]+`	nazwa zmiennej
`const`	`0|1`	stała (możliwe wartości początkowe zmiennej)
`=`	`=`	operator przypisania
`,`	`,`	separator
`:`	`:`	wskaźnik kubitu sterującego
`and( ( not(, or(, xor( )`	`and( ( not(, or(, xor( )`	otwarcie bramki and (not, or, xor)
`hdm( ( frd(, swp(, tfl( )`	`hdm( ( frd(, swp(, tfl( )`	otwarcie bramki Hadamarda (Fredkina, SWAP, Toffoliego)
`)`	`)`	nawias zamykający

5.2.3 Instrukcje i analiza semantyczna

Definiowanie zmiennej poprzez przypisanie jej wartości

$$\mathbf{E} \rightarrow^* \text{var} = \text{const}$$

Przypisanie wartości zmiennej jest równe jej deklaracji. Raz zadeklarowanej zmiennej nie można przypisać nowej wartości.

Na przykład `a = 0` jest deklaracją zmiennej o nazwie `a` z wartością równą 0.

Obliczenie funkcji boolowskiej na wcześniej zadeklarowanych zmiennych.

$$\mathbf{E} \rightarrow^* \text{var} = \mathbf{K}$$

Funkcję boolowską definiuje się poprzez bramki logiczne `and, or, xor, not`. Bramki `and, or, xor` są uogólnione do dowolnej liczby zmiennych. Instrukcja ta przypisuje wynik funkcji do zmiennej o nazwie *var* (leksemu odpowiadającemu temu tokenowi). Zmienna o nazwie *var* nie może być wcześniej zadeklarowana. Instrukcja ta nie zmienia wartości argumentów funkcji `and, or, xor, not`.

Na przykład instrukcja `f = and(a,or(c,not(d)))`, gdzie `a,b,c` to wcześniej zdefiniowane zmienne, przypisze wartość wyrażenia $\texttt{a} \wedge (\texttt{c} \vee \neg\texttt{d})$ do zmiennej o nazwie `f`.

Operowanie na zdefiniowanych zmiennych przy użyciu wybranych bramek kwantowych.

$$\mathbf{E} \rightarrow^* \mathbf{Q}$$

Bramki kwantowe operują na zadeklarownych zmiennych mutując ich stan. Nie zwracają żadnej wartości.

Dla bramek wielokubitowych, które przyjmują zmienne sterujące oraz wejściowe, zmienne sterujące oznaczane są poprzez poprzedzający ':'. Na przykład instrukacja `tfl(:a, b, :c)` oznacza bramkę Toffoliego na zmiennej wejściowej `b` ze zmiennymi strującymi `a`, `c`.

Powyższy język umożliwia na konstrukcje opisane w rodziale 4.

5.3 Translator

Opis zagadnienia translacji został wydzielony do rodziału 6.

5.4 Układ kwantowy z rejestrem

Przez układ kwantowy z rejestrem rozumiemy tutaj parę $(|\psi\rangle, G)$, gdzie $|\psi\rangle$ to stan wejściowy rejestru kubitów, rozumiany jako wektor, a G to ciąg kolejno nakładanych na rejestr bramek. Każda z bramek, na których użycie pozwala omawiany proagram, należy do jednego z poniższych typów:

`hdm` Bramka Hadamarda

`tfl` Uogólniona bramka Toffoliego

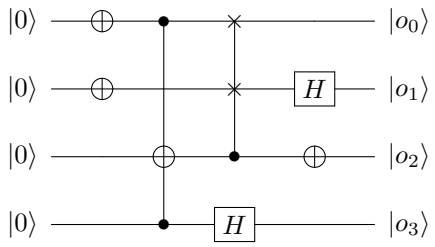
`frd` Bramka Fredkina

`swp` Bramka $SWAP$

Poza typem, każda bramka zawiera listę parametów. Dla bramki Hadamarda jest to indeks kubitu, w rejestrze kubitu wejściowego. Bramka Toffoliego jest paramatryzowana, poza indeksem kubitu wejściowego (z ang. *target*), przez listę ideksów kubitów sterujących (z ang. *conrol*). Bramka $SWAP$ jest paramatryzowana parą indeków kubitów wejściowych, a bramka Fredkina parą indeków kubitów wejściowych oraz indeksem kubitu sterującego.

5.4.1 Przykład

Następującemu układowi kwantowemu



odpowiada specyfikacja $(|\psi\rangle, G)$, gdzie

$$|\psi\rangle = |0000\rangle$$

$$G = [\ \texttt{not}(0),\ \texttt{not}(1),\ \texttt{tfl}([0,3])(2),\ \texttt{frd}(2)(0,1),\ \texttt{hdm}(1),\ \texttt{not}(2)\ ]$$

Porządek wykonywania bramek jest nadany od lewej do prawej. Dla bramek znajdujących się w tej samej kolumnie kolejność wykonywania jest nieistotna.

5.5 Symulator

Ciąg sperametryzowanych bramek przekształcana jest na ciąg macierzy odpowiadającym tym bramkom, a następnie przeprowadzana jest operacja mnożenia, która kondensuje listę przekształceń do pojedynczej macierzy M. W wyniku opracji monożenia macierzy M przez wektor stanu rejestru otrzymywany jest stan końcowy rejestru. Następnie stan ten jest "mierzony". To znaczy zwracany jest losowo, z prawdopodobieństwem wynikającym ze stanu rejestru, jeden z możliwych do zmierzenia stanów.

5.5.1 Generacja macierzy bramek z ich definicji

Żeby możliwe było mnożenie macierzy bramki przez stanu rejestru macierz ta musi mieć wymiary $2^n \times 2^n$ dla n kubitowego rejestru. Zatem wszystkie bramki operujące na podzbiorze kubitów z rejestru muszą być rozszerzone tak, by operować na całym rejestrze.

Bramki jednokubitowe

Niech M będzie macierzą bramki operujacej na jednym kubicie. Niech $|\psi\rangle$ będzie rejestrem n kubitów. Weźmy bramkę o macierzy M, która operuje na kubicie o indeksie $i < n$ (kubity numerowane od zera).

W celu obliczenia wyniku operacji tej bramki na stanie kwantowym $|\psi\rangle$ należałoby pomnożyć macierz M przez stan kubitu z indeksem i. Na stanach pozostałych kubitów należy przeprowadzić operację identyczności (której odpowiada macierz identyczności). Operacja łacząca stany kubitów oraz macierze operacji na pojedynczych kubitach to produkt tensorowy oznaczany $\otimes$.

Bramka kwantowa M po rozszerzeniu do n kubitów wygląda następująco:

$$M_n = Id^i \otimes M \otimes Id^{n-i-1}$$

gdzie przez Id^j rozumiemy produkt tensorowy j macierzy identyczności o wymiarach 2×2.

Bramki wielokubitowe

Analogicznie można rozszerzać bramki wielokubitowe jeśli operują na sąsiadujących w rejestrze kubitach. Jeżeli jednak bity sterujące i/lub bity wejściowe nie sąsiadują to można wykorzystać bramkę $SWAP$ do zamiany kolejności kubitów w rejetrze. Dzięki temu, że każdą permutację można rozbić na złożenie transpozycji, jest to zawsze możliwe do osiągnięcia.

Weźmy na przykład układ o rejetrze pięcio-kubitowym z bramką `tfl([0,3])(2)`. Układ ten został przestawiony na schemacie poniżej po lewej stronie. Diagram po prawej stronie jest równoważny temu układowi, ale wykorzystuje jedynie operacje na sąsiednich kubitach.

Zatem można rozpisać `tfl([0,3])(2)` = `swp(0,1)` $*$ `swp(2,3)` $*$ `tfl([0,3])(2)` $*$ `swp(0,1)` $*$ `swp(2,3)`. Wtedy jeśli każda z wykorzystywanych bramek zostanie rozszerzona, tak by operowała na pięciu kubitach, zgodnie ze strategią opisaną z rodziale 5.5.1, to zostanie wygenerowana macierz wykonująca operację `tfl([0,3])(2)` na rejestrze piecio-kubitowym.

Uogólniona bramka Toffoliego

Uogólniona bramka Toffoliego wykonuje operację negacji na bicie wejściowym gdy wszystkie bity strerujące są jedynkami.

Opisana w rodziale 5.5.1 bramka `tfl([0,3])(2)` operująca na pięcio-kubitowym rejestrze mapuje czyste stany kwantowe następująco:

$$10010 \rightarrow 10110 \text{ i } 10110 \rightarrow 10010$$

$$11010 \rightarrow 11110 \text{ i } 11010 \rightarrow 11110$$

$$10011 \rightarrow 10111 \text{ i } 10011 \rightarrow 10111$$

$$11011 \rightarrow 11111 \text{ i } 11011 \rightarrow 11111$$

w pozostałych przypadkach dokonuje mapowania identycznościowego.

Bramka Toffoliego jest macierzą permutacji, która w każdym wierszu oraz kolumnie ma dokładnie jedną jedynką oraz $n-1$ zer. Dla mapowania

$$10010 \rightarrow 10110$$

jedynka powinna się znaleźć w 19 kolumnie, 23 wierszu, ponieważ

$$10010_{(2)} = 18$$

$$10110_{(2)} = 22$$

gdzie numaracja zaczyna się od zera.

Pseudokod 5.1: Generacja macierzy bramki Toffoliego z defninicji bramki

```
Input: Liczba kubitów w rejestrze n, zbiór indeksów bitów sterujących C, indeks bitu wejściowego i
Output: Macierz bramki Toffoliego M
1  M ← macierz wymiarów 2^n × 2^n wypełniona 0
2  for j ← 0 to 2^n do
3      isId ← false
4      foreach c ∈ C do
           /* bity w zapisie biranym numerujemy od najmiej znaczącego, indeksem 0 */
5          if j ma 0 na (n − c − 1)-tym bicie w zapisie binarnym then
6              isId ← true
7      if idId then
           /* przypadek gdy, wszystkie kubity sterujące są jedynkami */
8          wstaw 1 do macierzy M na miejscu (j, j bitwise_not na (n − i)-tym bicie) /* (kolumna, wiersz) */
9      else
10         wstaw 1 do macierzy M na miejscu (j, j)
```

5.6 Wyjście

Program zwraca schemat układu kwantowego, który został stworzony w wyniku działania translatora. Układ ten odpowiada obliczeniom zdefiniowanym w opisie wejściowym.

Poza układem program zwraca wynik obliczeń, czyli końcowy, zmierzony, stan rejestru kwantowego po symulacji działania układu kwantowego. Dodatkowo zwaraca informację o prawdopodobienstwie otrzymania stanu $|1\rangle$ dla każdego z kubitów w rejestrze.

Translacja układów logicznych do układów kwantowych

Jednym z najważniejszych zagadnień, którymi zajmuje się ta praca, jest translacja układów logicznych, rozumianych jako zestaw funkcji boolowskich (rodział 2), do układów kwantowych.

6.1 Problem translacji

W przeciwieństwie do bramek logicznych, operacje składające się na układ kwantowy muszą być unitarne. W szczególności znaczy to, że przyjmują na wejściu tyle samo bitów ile zwracają na wyjściu. Zatem chcąc wyrazić te same operacje, które wykonuje układ logiczny za pomocą układu kwantowego, traktowanego tutaj jako pojedyncza operacja, należałoby odpowiednio zwiększyć liczbę bitów na wejściu/wyjściu i następnie zignorować te "dodatkowe"bity na wejściu/wyjściu. Nie zapewnia to jednak, że tak stworzone mapowanie przedstawione jako macierz będzie poprawną bramkę kwantową (tzn. czy macierz ta będzie unitarna).

Weźmy na przykład bramkę układ logiczny składający się z pojedynczej bramki AND. Żeby zachować stałą liczbę bitów na wejściu i wyjściu możemy stworzyć funkcję:

$$f_{AND}(a,b) = (0, ab)$$

Tak stworzona fukcja nie jest nawet odwracalna, co jest warunkiem koniecznym unitarności. Co więcej nie jestnieje taka funckcja boolowska h

$$f_{AND}(a,b) = (h(a,b), ab)$$

żeby f_{AND} było funkcją odwracalną.

Dlatego symulowanie układów logicznych za pomocą układów kwantowych wymaga czasami wprowadzenia dodatków bitów, poza oczywistym rozszerzeniem wyjścia lub wejścia, tak by były równoliczne.

6.2 Podejście naiwne

Jeżeli jednak stworzony układ kwantowy nie jest pojedynczą bramką, a ciągiem operacji, to każda z tych operacji musi być uniratna. Oznacza to w szczególności, że nie tylko liczbą bitów na wejściu do układu i wyjściu z układu muszą być sobie równe, ale że liczba bitów, na których bramki kwantowe operują jest stała. W szczególności nie można dokonać żodzielenia kabla"jak to się dzieje w przypadku układów klasycznych.

Przykładowym rozwiązaniem tego problemu mogłoby być wprowadzenie rozszerzenie rejestru kubitów, tak by każdej zmiennej odpowiadała Dlatego o każdej takiej opracji będziemy myśleć jako o bramce $COPY$. Której zachowanie można zapisać za pomocą funkcji następująco:

$$COPY(a) = (a,a)$$

Jeżeli potrafimy wyrazić bramkę $COPY$ oraz zestaw bramek logicznych tworzących układ funkcjonalnie zupełny za pomocą bramek kwantowych to potrafimy wyrazić każdy układ logiczny za pomocą układu kwantowego.

Zatem naturalnie pierwszym podejściem do problemu translacji układów logicznych na kwantowe jest wyrażenie tego układu używając jedynie pewnego ograniczonego (zupełnego) zestawu bramek, na przykład $COPY, NOT, AND$, a następnie bezpośrednia zamiana każdej z tych operacji na bramki kwantowe.

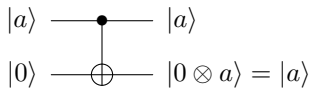
6.2.1 Copy

Ze względu na stałą liczbę kubitów w układzie kwantowym, każda bramka $COPY$ wymaga powiększenia rejestru wejściowego o jeden bit. Zatem "kwantowa bramka $COPY$"będzie dokonywała mapowania

$$C(a,0) = (a,a)$$

dla $a \in \{0,1\}$.

Operację te można wyrazić za pomocą bramki Feymana



6.2.2 Not

Logicznej opracji NOT odpowiada bramka kwantowa NOT.

6.2.3 And

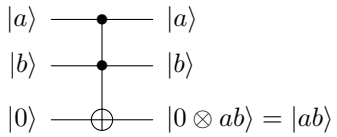
Weźmy funkcję

$$T(a,b,c) = (a,b,c \otimes ab)$$

Wtedy jeżeli $c = 0$ to

$$T(a,b,0) = (a,b,ab)$$

Operacji tej odpowiada bramka Toffoliego:



Zatem po wprowadzeniu dodatkowego bitu 0, można obliczyć AND za pomocą bramki Toffoliego.

6.3 Wyrażanie funkcji boolowskich za pomocą układów kwantowych

6.3.1 Wejście

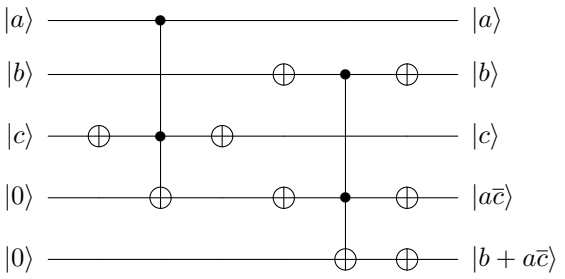
W tej pracy, zgodnie z modelem przedstawionym w rozdziale 2, myślimy jako o wektorze funkcji boolowskich. Zatem wejściem do translatora jest funkcja boolowska, ponieważ będzimy zajmować się "jedną na raz". Dodatkowo zakładamy, że każda zadeklarowana zmienna jest również częścią wyjścia, zatem nie możemy źatracić"wartości tych zmiennych.

6.3.2 Przykład

Weźmy przykładową funkcję

$$f(a,b,c) = b + a\bar{c} = b \vee (a \wedge (\neg c))$$

Korzystając z metod translacji opisanych w rozdziale 6.2, można przestawić tę funkcję w następujący sposób:



Na wejściu zostały dodane dwa bity. Używając strategii tłumaczenia każdej operacji AND, OR, NOT na zestaw bramek kwantowych liczba dodatkowych kubitów na wejściu rośnie z sumaryczną liczbą bramek OR i AND. Jeden bit na wejściu jest konieczny to przechowania wyniku funkcji, ale wartości przechowywany na reszcie z dodatkowych kubitów są niestotne (z ang. garbage). Kubity są zasobami ograniczonymi, których użycie warto minimalizować i zastanowić się jak inaczej podejść do problemu tanslacji.

6.3.3 Teoretyczne minimum

Lemat 6.1 *Z każdej funkcji boolowskiej $f(\vec{x}) = f(x_0, x_1, \dots x_n)$, dla pewnego $n \in \mathbb{N}$ oraz $\vec{x} \in \{0,1\}^n$, można stworzyć funkcję $f_Q : \{0,1\}^{n+1} \to \{0,1\}^{n+1}$, dla której*

$$f_Q(x_0, x_1, \dots x_n, 0) = (x_0, x_1, \dots x_n, f(x_0, x_1, \dots x_n))$$

oraz, której odpowiada macierz unitarna wymiarów $2^{n+1} \times 2^{n+1}$.

Dowód. Zdefiniujmy f_Q następująco

$$f_Q(x_0, x_1, \dots x_n, c) = (x_0, x_1, \dots x_n, c \otimes f(\vec{x}))$$

Dla c = 0 mamy wtedy

$$f_Q(x_0, x_1, \dots x_n, 0) = (x_0, x_1, \dots x_n, 0 \otimes f(\vec{x}))$$

Dla tak zdefiniowanej funkcji f_Q mamy następujące mapowanie

$$|x'_0, x'_1, \dots x'_n, 0\rangle \to \left|x'_0, x'_1, \dots x'_n, f(\vec{x'})\right\rangle = |x'_0, x'_1, \dots x'_n, 0\rangle$$

$$|x'_0, x'_1, \dots x'_n, 1\rangle \to \left|x'_0, x'_1, \dots x'_n, 1 \otimes f(\vec{x'})\right\rangle = |x'_0, x'_1, \dots x'_n, 1\rangle$$

dla argumentów $\vec{x'}$, że $f(\vec{x'}) = 0$.
Analogicznie

$$|x''_0, x''_1, \dots x''_n, 0\rangle \to \left|x''_0, x''_1, \dots x''_n, f(\vec{x''})\right\rangle = |x''_0, x''_1, \dots x''_n, 1\rangle$$

$$|x''_0, x''_1, \dots x''_n, 1\rangle \to \left|x''_0, x''_1, \dots x''_n, 1 \otimes f(\vec{x''})\right\rangle = |x''_0, x''_1, \dots x''_n, 0\rangle$$

dla argumentów $\vec{x''}$, że $f(\vec{x''}) = 1$.
Wtedy widać, że

$$f_Q(f_Q(x_0, x_1, \dots x_n, c)) = (x_0, x_1, \dots x_n, c)$$

Zatem dla odpowiadającej powyższemu mapowaniu macierzy permutacji M mamy, że $\forall x_0, x_1, \dots x_n, \in \{0,1\}$

$$MM\,|x_0, x_1, \dots x_n\rangle = |x_0, x_1, \dots x_n\rangle$$

Stąd

$$MM = I$$

♦

Stąd widać, że każda funkcję boolowską można zamienić na bramkę kwantową z użyciem maksymalnie jednego dodatkowego kubitu. Chociaż jest to wynik ważny to nie bardzo użyteczny z praktycznego punktu widzenia, jeżeli nie portafimy rozbić tej bramki na serię bardziej uniwersalnych bramek, tak by móc symulować każdą funkcję boolowska za pomocą niewielkiego zestawu bramek kwantowych.

6.4 Postać ESOP

W rozdziale 2 omówiona jest sumacyjna postać kanoniczna fukcji boolowskiej. Funkcja w tej postaci jest wyrażona jako suma produktów (termów). Analogiczną postacią jest postać ESOP czyli xor termów.

Twierdzenie 6.1 *Każdą funkcję boolowską można zapisać w postaci ESOP.*

Dowód. Weźmy funkcję boolowską $f : \{0,1\}^n \to \{0,1\}$ zapisaną w postaci sumy mintermów.

$$f(a_0, a_1, \ldots a_n) = m_{k_0} + m_{k_1} + \ldots + m_{k_l}$$

gdzie m_i oznacza minterm, a $k_0, k_1, k_l \in \{0, 1, \ldots n\}$ oznaczają indeksy mintermów należących do funkcji f. Zauważmy następującą tożsamość

$$a + b = a \otimes b \otimes ab$$

Wtedy, ponieważ $\forall i,j$

$$m_i \otimes m_j = 0$$

funkcję f można zapisać

$$f(a_0, a_1, \ldots a_n) = m_{k_0} \otimes m_{k_1} \otimes \ldots \otimes m_{k_l}$$

♦

6.5 Tworzenie układu kwantowego z postaci ESOP

Twierdzenie 6.2 *Każdą funkcję boolowską $f : \{0,1\}^n \to \{0,1\}$ można obliczyć za pomocą układu o rejestrze z $n+1$ kubitami składającej się z maksymalnie 2^{n-1} uogólnionych bramek Toffoliego oraz $2^n * n + 1$ bramek NOT.*

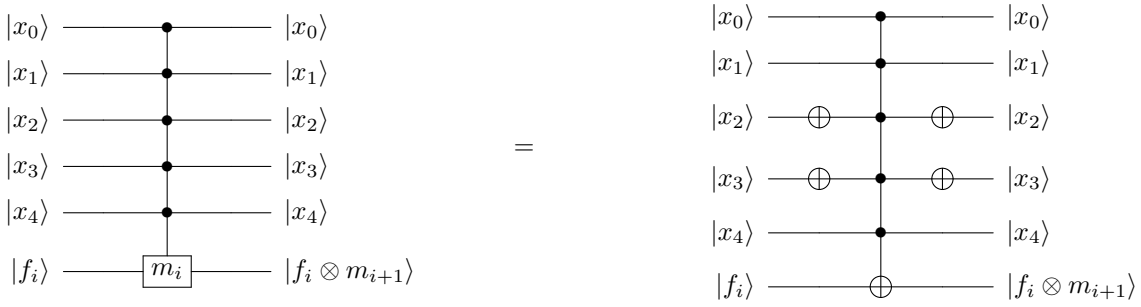
Dowód. Weźmy funkcję f w postaci ESOP

$$f(a_0, a_1, \ldots a_n) = m_{k_0} \otimes m_{k_1} \otimes \ldots \otimes m_{k_l}$$

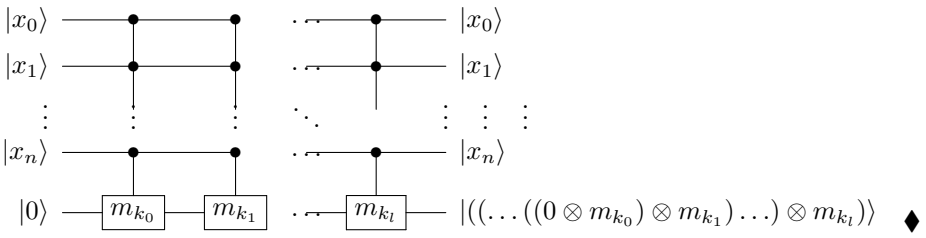
Wtedy każdy minterm można obliczyć za pomocą bramki Toffoliego oraz maksymalnie $2 * n$ NOT.
Mając na ostatniej linii wynik operacji xor pierwszych $i < l$ minterów należacych do funkcji f

$$f_i = m_{k_0} \otimes m_{k_1} \otimes \ldots \otimes m_{k_i}$$

Można 'rozszerzyć wynik' o kolejny minterm w następujący sposób, że dla przykładowego mintermu $m_{i+1} = x_0 x_1 \overline{x_2} \overline{x_3} x_4$ tworzymy następujący układ kwantowy



Wtedy można skonstruować następujący układ kwantowy:



6.5.1 Algorytm

W celu optymalizacji liczby wykorzystanych bramek `not` zapisania funkcji za pomocą bramek kwantowych algorytm przetrzymuje aktualny stan zmiennych. To znaczy czy kubit odpowiadający danej zmiennej przechowuje jej wartość czy negację tej wartości. Wykorzystuje do tego zbiór, który zawiera zmienne, których negacja jest aktualnie przechowywana w rejestrze.

Pseudokod 6.1: Konwersja postaci ESOP to listy bramek kwantowych

```
Input: Zbiór termów funckcji w postaci ESOP E, index wyjścia o
Output: Lista bramek kwantowych G
negPol ← pusta lista
G ← pusta lista
foreach t ∈ E do
    /* v to para index oraz informacja o polaryzacji */
    foreach v ∈ t do
        if v postaci x̄, gdzie x to zmienna then
            if v.id ∉ negPol then
                do G dodaj not(v.id)
                do negPol dodaj v.id
        else
            if v.id ∈ negPol then
                do G dodaj not(v.id)
                z negPol usuń v.id
    do G dodaj tfl(id elementów z t)(o)
/* przywrócenie stanu początkowego */
foreach i ∈ negPol do
    do G dodaj not(i)
```

6.6 Rozwinięcie Shannona

Ponieważ celem jest translacja układów kwantowych na układy kwantowe to funkcje boolowskie na wejściu zostają zdefiniowane przez klasyczne bramki logiczne, czyli za pomocą operacji AND, NOT, OR oraz XOR. Tak zdefiniowane funkcje chcemy zmienić na postać ESOP. Najprostszym podejściem mogłoby być wyliczenie funkcji f dla każdego możliwego wartościowania argumentów, otrzymując tym samym zbiór mintermów należacych do funkcji. Czasami jednak to podejście może nie być najbardziej optymalne.

Weźmy następująco zdefiniowaną funkcję

$$f(\vec{x}) = and(x_0, h(\vec{x}))$$

gdzie $\vec{x}$ jest wektorem długości $n \in \mathbb{N}$ oraz $n >> 1$, a h jest 'skomplikowaną' funkcją boolowską.
Wtedy połowa mintermów, gdzie $x_0 = 0$ będzie dawało fałsz i nie warto ich liczyć. Lepszą alternatywą może być skorzysanie z rozwinięć funkcji logicznej.

Definicja 6.1 *Rozwinięcie Shannona dla funkcji boolowskiej f względem zmiennej x_i denifiuje się następująco*

$$f(x_0, x_1, \dots x_n) = x_i * f_{x_i} \otimes \overline{x_i} * f_{\overline{x_i}}$$

gdzie

$$f_{x_i} = f(x_i = 1)$$
$$f_{\overline{x_i}} = f(x_i = 0)$$

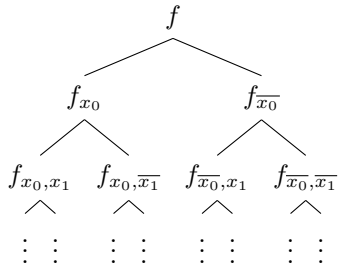
To znaczy, f_{x_i} jest funkcją powstałą przez podstawienie za x_i 1.

6.6.1 Algorytm

Korzystając z rozwinięć Shannona budujemy drzewo w następujący sposób:

1. Podstaw pod korzeń funkcję f. Niech $f_{curr} := f$.
2. Dla funkcji f_{curr} jeśli nie jest funkcją stała to $a :=$ dowolny z argumentów wejściowych funkcji f_{curr} i podstaw jako prawego syna f_a, a jako lewego $f_{\overline{a}}$.
3. Wykonaj 2. dla synów f_{curr}.

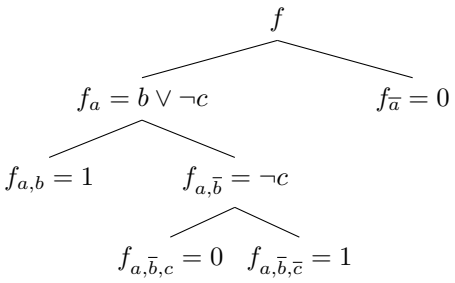
Wynikiem tego algorytmu jest następująco wyglądające drzewo:



Na podstawie tak zbudowanego drzewa można łatwo znaleźć postać ESOP funkcji f. Każdej funkcji f_l będącej liściem, zatem funkcją stałą odpowiada term, na przykład $f_{x_0,\overline{x_1},x_2}$ odpowiada term $x_0\overline{x_1}x_2$. Jeżeli liść jest równy 1 to dany term należy do funkcji.

Przykład

Dla przykładowej funkcji $f(a,b,c) = a \wedge (b \vee \neg c)$ algorytm stworzy następujące drzewo:



Stąd

$$f(a, b, c) = ab \otimes a\overline{b}\overline{c}$$

Implementacja systemu

7.1 Opis technologii

Do napisania programu został wykorzystany język Scala w wersji 2.12.8. Do operacji na macierzach została wykorzystana bibiolteka Breeze [1]. Do napisania parsera została wykorzystana natywna biblioteka Scali scala-parser-combinators. [2].

7.2 Instrukcja obługi

Warunkiem wstępnym do skorzystania z programu jest posiadanie zainstalowane Java JDK przynajmniej w wersji 8.

[1] https://github.com/scalanlp/breeze
[2] https://github.com/scala/scala-parser-combinators

Przykłady użycia

Podsumowanie

Bibliografia

[1] K. Fazel, M. Thornton, J. Rice. Esop-based toffoli gate cascade generation. strony 206 – 209, 09 2007.

[2] J. Hui. Quantum computing series. Strona: https://medium.com/@jonathan˙hui/qc-quantum-computing-series-10ddd7977abd, 2017.

[3] A. Ligęza. Elementy logiki dla informatyków, wykład iii, elementy logiki. algebra boole'a. analiza i synteza układów logicznych. Strona: https://ai.ia.agh.edu.pl/˙media/pl:dydaktyka:logic:logika-boole-synthesis-9.pdf.

[4] A. Muthukrishnan. Classical and quantum logic gates: An introduction to quantum computing. 1999.

[5] M. A. Nielsen, I. L. Chuang. *Quantum Computation and Quantum Information, 10th Anniversary Edition.* Cambridge University Press, 2010.

[6] N. Song, M. Perkowski. Exorcism-mv-2: Minimization of exclusive sum of products expressions for multiple-valued input incompletely specified functions. strony 132–137, 01 1993.

[7] D. Voudouris, M. Sampson, G. Papakonstantinou. Variable reordering for reversible wave cascades.

Uogólniona bramka Toffoliego