

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

1	Wstęp	1
2	Logika klasyczna	3
2.1	Funkcja boolowska	3
2.1.1	Podstawowe funkcje boolowskie	3
2.1.2	Kanoniczna postać sumacyjna (SOP)	4
2.2	Układy logiczne	4
3	Model układu kwantowego	7
3.1	Stan kwantowy kubit	7
3.1.1	Notacja Diraca	7
3.2	Stan kwantowy rejestru kubitów	7
3.2.1	Stan rejestru kubitów a stanów kubitów	8
3.2.2	Splątanie kwantowe	8
3.3	Bramka kwantowa	8
3.4	Wybrane bramki kwantowe	8
3.4.1	Bramka NOT	8
3.4.2	Uogólniona bramka Toffoliego	9
3.4.3	Bramka SWAP	10
3.4.4	Bramka Fredkina	10
3.4.5	Bramka Hadamarda	10
3.5	Układ kwantowy	11
4	Projekt systemu	13
4.1	Wymagania	13
4.1.1	Założenia dotyczące opisu wejściowego	13
4.1.2	Założenia dotyczące zwracanego wyniku	13
4.1.3	Wymagania dodatkowe	13
4.2	Architektura Systemu	13
4.3	Język wejściowy i Parser	14
4.3.1	Gramatyka	14
4.3.2	Analiza leksykalna	14
4.3.3	Instrukcje i analiza semantyczna	14
4.4	Translator	15
4.5	Układ kwantowy z rejestrem	15
4.5.1	Przykład	15
4.6	Symulator	16
4.6.1	Generacja macierzy bramek z ich definicji	16
4.7	Wyjście	17
5	Translacja układów logicznych do układów kwantowych	19
5.1	Problem translacji	19
5.2	Podejście naiwne	19
5.2.1	Copy	20

5.2.2	Not	20
5.2.3	And	20
5.3	Wyrażanie funkcji boolowskich za pomocą układów kwantowych	20
5.3.1	Wejście	20
5.3.2	Przykład	21
5.3.3	Teoretyczne minimum dla bitów pomocniczych	21
5.4	Postać ESOP	21
5.5	Tworzenie układu kwantowego z postaci ESOP	22
5.5.1	Przykład	23
5.5.2	Algorytm	23
5.6	Rozwinięcie Shannona	23
5.6.1	Algorytm	24
6	Implementacja systemu	27
6.1	Opis technologii	27
6.2	Instrukcja obsługi	27
6.3	Język wyjścia	27
7	Przykłady użycia	29
7.1	Funkcja boolowska	29
7.1.1	Wejście	29
7.1.2	Wyjście	29
7.2	Układ logiczny z wartościami 1/2	29
7.2.1	Wejście	29
7.2.2	Wyjście	29
7.3	Bramki kwantowe	30
7.3.1	Wejście	30
7.3.2	Wyjście	30
8	Podsumowanie	31
8.1	Minimalizacja liczby kubitów pomocniczych	31
8.1.1	Wnioski i uwagi	31
	Bibliografia	33

Wstęp

W latach 70. po raz pierwszy zostało użyte sformułowanie "kwantowej teorii informacji" uregulujące użycie efektów kwantowych do manipulacji informacją. Niedługo potem pojawiła się idea układów kwantowych, które analogicznie do układów logicznych mają pozwalać na przeprowadzanie obliczeń. W 1994 temat komputerów kwantowych stał się bardzo interesujący, gdy Peter Shor opublikował algorytm wykorzystujący układy kwantowe, który rozwiązuje problem faktoryzacji 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 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 rozwiązanymi 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 kwantowych, które będą dalej używane w tej pracy.

Rozdział 4 zawiera wymagania programu systemu, schemat architektury oraz opisy poszczególnych komponentów.

Do rozdziału 5 został wydzielony opis zagadnienia translacji.

W rozdziale 6 są omówione szczegóły implementacyjne. Opisane są użyte technologie oraz sposób użycia.

Ostatni rozdział 7 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

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

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)

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

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

x	$f(x) = \bar{x}$
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:

$$x^e = \begin{cases} x & \text{gdzie } e = 1 \\ \bar{x} & \text{gdzie } e = 0 \end{cases}$$

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

Definicja 2.4 Term to produkt literalów (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

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

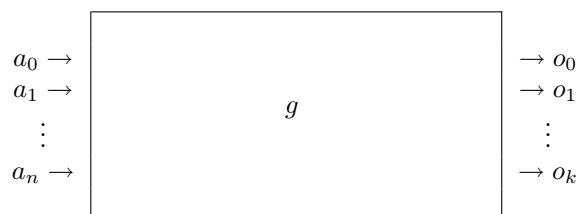
gdzie $b_i \in \{0,1\}$ jest wskaźnikiem (mówi czy dany minterm należy do funkcji f), a m_i to minterm, taki że

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

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

2.2 Układy logiczne

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





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

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

gdzie f_0, f_1, \dots, f_k to funkcje boolowskie.



Model układu kwantowego

3.1 Stan kwantowy kubit

Podstawowym nośnikiem informacji w obliczeniach kwantowych, analogicznym do klasycznego bitu, jest bit kwantowy nazywany kubit. 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.

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

Stan kubit 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 prawdopodobieństwie 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 $|\psi\rangle$ 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 kubit odpowiada znormalizowanemu wektorowi w 2-wymiarowej przestrzeni Hilberta.

3.2 Stan kwantowy rejestru kubitów

Stan n-kubitowy rejestru kwantowego to

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

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

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



3.2.1 Stan rejestru 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 $|q_0\rangle, |q_1\rangle, \dots, |q_n\rangle$

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

3.2.2 Splątanie kwantowe

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

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

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:

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

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 następujący 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

$$|a\rangle \longrightarrow \oplus \longrightarrow |\neg a\rangle$$

3.4.2 Uogólniona bramka Toffoliego

Bramka Feymana

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

$$|00\rangle \rightarrow |00\rangle$$

$$|01\rangle \rightarrow |01\rangle$$

$$|10\rangle \rightarrow |11\rangle$$

$$|11\rangle \rightarrow |10\rangle$$

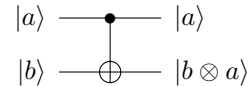
Inaczej jej działanie można zapisać w postaci

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

Wtedy można patrzeć na nią jak na uogólnioną bramkę *XOR*.

Bramce tej odpowiadają następująca macierz i zapis w układzie kwantowym:

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



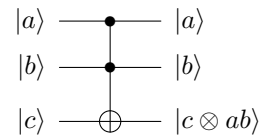
Bramka Toffoliego

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 kontrolującymi $C_n NOT$.

$$C_n NOT |a_0, a_1, \dots, a_n\rangle = C_n NOT |a_0, a_1, \dots, a_n \otimes a_0 a_1 \dots a_{n-1}\rangle$$

Wtedy dla $n = 2$ mamy bramkę Toffoliego, dla $n = 1$ bramkę Feynmana, 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:

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{array}{cc} |a\rangle & \text{---}\times\text{---} |b\rangle \\ |b\rangle & \text{---}\times\text{---} |a\rangle \end{array}$$

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 układu kwantowego dla tej bramki to:

$$CSWAP = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{array}{cc} |a\rangle & \text{---}\bullet\text{---} |a\rangle \\ |b\rangle & \text{---}\times\text{---} |b'\rangle \\ |c\rangle & \text{---}\times\text{---} |c'\rangle \end{array}$$

3.4.5 Bramka Hadamarda

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

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = |+\rangle \\ H|1\rangle &= \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle = |-\rangle \end{aligned}$$

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

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

$$|a\rangle \longrightarrow \boxed{H} \longrightarrow H|a\rangle$$

3.5 Układ kwantowy

Definicja 3.1 Przez układ kwantowy rozumiemy ciąg bramek kwantowych, które są zdefiniowane jako operacje na n -bitowym rejestrze 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.



Projekt systemu

4.1 Wymagania

Celem tej pracy jest stworzenie programu, który przyjmuje opis obliczeń (używając predefiniowanego języka) dla układu kwantowego. Następnie tworzy układ kwantowy odpowiadaj 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

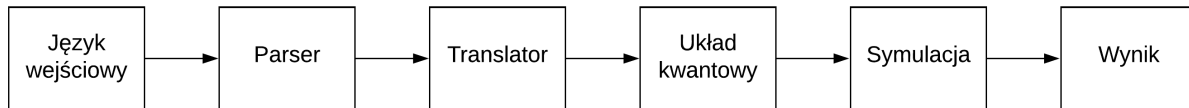
1. Układ kwantowy w postaci stanu początkowego rejestru oraz serii bramek.
2. Wynik symulacji.

4.1.3 Wymagania dodatkowe

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

4.2 Architektura Systemu

Na rysunku 4.1 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 4.1: Diagram potoku

4.3 Język wejściowy i Parser

4.3.1 Gramatyka

$P \rightarrow EP \mid E$
 $E \rightarrow \text{var} = V \mid Q$
 $V \rightarrow \text{var} \mid \text{const} \mid K$
 $W \rightarrow V, W \mid V$
 $K \rightarrow \text{and}(W) \mid \text{not}(V) \mid \text{or}(W) \mid \text{xor}(W)$
 $Q \rightarrow \text{hdm}(\text{var}) \mid \text{swp}(\text{var}, \text{var}) \mid \text{tfl}(L \text{ var } R) \mid F$
 $F \rightarrow \text{frd}(: \text{var}, \text{var}, \text{var}) \mid \text{frd}(\text{var}, : \text{var}, \text{var}) \mid \text{frd}(\text{var}, \text{var}, : \text{var})$
 $L \rightarrow L : \text{var}, \mid \epsilon$
 $R \rightarrow , : \text{var } R \mid \epsilon$

4.3.2 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

4.3.3 Instrukcje i analiza semantyczna

Definiowanie zmiennej poprzez przypisanie jej wartości

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

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.

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

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

Operowanie na zdefiniowanych zmiennych przy użyciu wybranych bramek kwantowych.

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

Bramki kwantowe operują na zadeklarowanych 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 instrukcja `tfl(:a, b, :c)` oznacza bramkę Toffoliego na zmiennej wejściowej b ze zmiennymi sterującymi a, c .

Język ten spełnia wcześniej opisane wymagania.

4.4 Translator

Opis zagadnienia translacji został wydzielony do rozdziału 5.

4.5 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 program, należy do jednego z poniższych typów:

`hdm` Bramka Hadamarda

`tfl` Uogólniona bramka Toffoliego

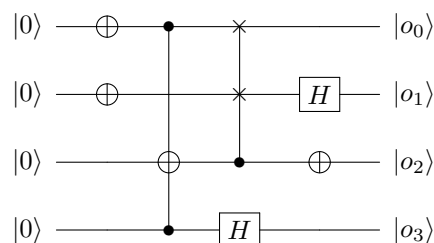
`frd` Bramka Fredkina

`swp` Bramka *SWAP*

Poza typem, każda bramka zawiera listę parametrów. Dla bramki Hadamarda jest to indeks kubitów, w rejestrze kubitów wejściowego. Bramka Toffoliego jest parametryzowana, poza indeksem kubitów wejściowego (z ang. *target*), przez listę indeksów kubitów sterujących (z ang. *control*). Bramka *SWAP* jest parametryzowana parą indeksów kubitów wejściowych, a bramka Fredkina parą indeksów kubitów wejściowych oraz indeksem kubitów sterującego.

4.5.1 Przykład

Następującemu układowi kwantowemu



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

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

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

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



4.6 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 operacji mnożenia 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.

4.6.1 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 operującej 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 kubitów z indeksem i . Na stanach pozostałych kubitów należy przeprowadzić operację identyczności (której odpowiada macierz identyczności). Operacja łącząca 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 rejestrze. Dzięki temu, że każdą permutację można rozbić na złożenie transpozycji, jest to zawsze możliwe do osiągnięcia.

Weźmy na przykład układ o rejestrze pięcio-kubitowym z bramką $\text{tf1}([0,3])(2)$. Układ ten został przedstawiony na schemacie poniżej po lewej stronie. Diagram po prawej stronie jest równoważny temu układowi, ale wykorzystuje jedynie operacje na sąsiednich kubitach.



Zatem można rozpisać $\text{tf1}([0,3])(2) = \text{swp}(0,1) * \text{swp}(2,3) * \text{tf1}([0,3])(2) * \text{swp}(0,1) * \text{swp}(2,3)$. Wtedy jeśli każda z wykorzystywanych bramek zostanie rozszerzona, tak by operowała na pięciu kubitach, zgodnie ze strategią opisaną z rozdziale 4.6.1, to zostanie wygenerowana macierz wykonująca operację $\text{tf1}([0,3])(2)$ na rejestrze pięcio-kubitowym.

Uogólniona bramka Toffoliego

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

Opisana w rozdziale 4.6.1 bramka $\text{tf1}([0,3])$ (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 jedynkę 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 4.1: Generacja macierzy bramki Toffoliego z definicji bramki

Input: Liczba kubitów w rejestrze n , zbiór indeksów bitów sterujących C , indeks bitu wejściowego i

Output: Macierz bramki Toffoliego M

```
1  $M \leftarrow$  macierz wymiarów  $2^n \times 2^n$  wypełniona 0;
2 for  $j \leftarrow 0$  to  $2^n$  do
3    $isId \leftarrow \text{false}$ ;
4   foreach  $c \in C$  do
5     /* bity w zapisie binarnym numerujemy od najmniej znaczącego, indeksem 0 */
6     if  $j$  ma 0 na  $(n - c - 1)$ -tym bicie w zapisie binarnym then
7        $isId \leftarrow \text{true}$ ;
8   if  $isId$  then
9     /* przypadek gdy, wszystkie kubity sterujące są jedynkami */
10    wstaw 1 do macierzy  $M$  na miejscu  $(j, j \text{ bitwise\_not na } (n - i)\text{-tym bicie})$ ; // (kolumna, wiersz)
11  else
12    wstaw 1 do macierzy  $M$  na miejscu  $(j, j)$ ;
```

4.7 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 zwraca informację o prawdopodobieństwie 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 (rozdział 2), do układów kwantowych. Celem przedstawionego tu problemu translacji jest stworzenie układu kwantowego, który oblicza te same funkcje boolowskie co zadany układ, z wykorzystaniem zestawu bramek przedstawionego w wcześniejszych rozdziałach, czyli bramek należących do rodziny bramek Toffoliego, bramki *SWAP* oraz bramki Fredkina.

5.1 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 bramką 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 funkcja nie jest nawet odwracalna, co jest warunkiem koniecznym unitarności. Co więcej nie jest to taka funkcja 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 bitów pomocniczych, które stanowią dodatkową przestrzeń do obliczeń i których wartość jest początkowa jest znana, tutaj zawsze $|0\rangle$.

5.2 Podejście naiwne

Jeżeli jednak stworzony układ kwantowy nie jest pojedynczą bramką, a ciągiem operacji, to każda z tych operacji musi być unitarna. 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 "rozdzielenia kabla" jak to się dzieje w przypadku układów klasycznych.

Przykładowym rozwiązaniem tego problemu mogłoby być wprowadzenie rozszerzenia rejestru kubitów, tak by każdej zmiennej odpowiadała Dłatego o każdej takiej operacji 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.

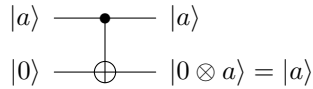
5.2.1 Copy

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 tę można wyrazić za pomocą bramki Feynmana



5.2.2 Not

Logicznej operacji *NOT* odpowiada bramka kwantowa *NOT*.

5.2.3 And

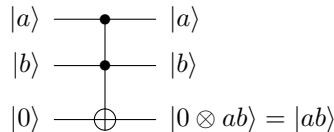
Weźmy funkcję

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

Wtedy jeżeli $c = 0$ to

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

Operacji tej odpowiada bramka Toffoliego:



Zatem po wprowadzeniu bitu pomocniczego, można obliczyć *AND* za pomocą bramki Toffoliego.

5.3 Wyrażanie funkcji boolowskich za pomocą układów kwantowych

5.3.1 Wejście

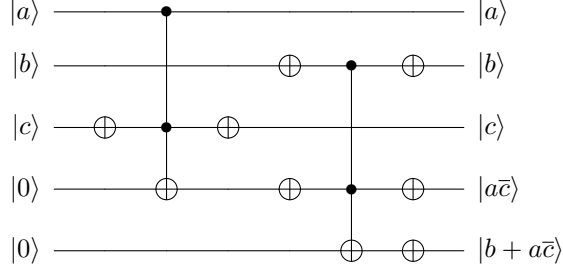
W tej pracy, zgodnie z modelem przedstawionym w rozdziale 2, myślimy jako o wektorze funkcji boolowskich. Zatem wejściem do translatora jest funkcja boolowska, ponieważ będziemy zajmować się "jedną na raz". Dodatkowo zakładamy, że każda zadeklarowana zmienna jest również częścią wyjścia, zatem nie możemy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

5.3.2 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 5.2, 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 potrzebnych kubitów pomocniczych u rośnie z sumaryczną liczbą bramek *OR* i *AND*. Jeden bit na wejściu jest konieczny to przechowania wyniku funkcji, ale wartości przechowywane na reszcie z kubitów pomocniczych są niestotne.

5.3.3 Teoretyczne minimum dla bitów pomocniczych

Lemat 5.1 *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 $f_Q : \{0, 1\}^{n+1} \rightarrow \{0, 1\}^{n+1}$, 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. Zdefiniujemy f_Q następująco

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

Wtedy dla $c = 0$

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

Funkcją przeciwną do f_Q jest ona sama.

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

Ponieważ funkcja f_Q jest bijekcją można wyrazić ją jako permutację na wektorach $\{0, 1\}^{n+1}$. Dla macierzy M wyrażającej tę permutację otrzymujemy $\forall x_0, x_1, \dots, x_n \in \{0, 1\}^{n+1}$

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

Stąd

$$MM = I$$

◆

Z lematu 5.1 wynika, że każdą funkcję boolowską można wyrazić za pomocą układu kwantowego z użyciem tylko jednego bitu pomocniczego, który na wyjściu przechowuje wynik tej funkcji.

5.4 Postać ESOP

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



Twierdzenie 5.1 Każdą funkcję boolowską można zapisać w postaci ESOP.

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

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

gdzie m_i oznacza minterm, a $k_0, k_1, k_l \in \{0, 1, \dots, n\}$ 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

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

◆

5.5 Tworzenie układu kwantowego z postaci ESOP

Twierdzenie 5.2 Każdą funkcję boolowską $f : \{0, 1\}^n \rightarrow \{0, 1\}$ można obliczyć za pomocą układu, o rejestrze $n + 1$ -kubitowym, składającego się z maksymalnie 2^n uogólnionych bramek Toffoliego oraz $2^{n+1} * n$ bramek NOT .

Dowód. Weźmy funkcję f w postaci alternatywy wykluczającej mintermów.

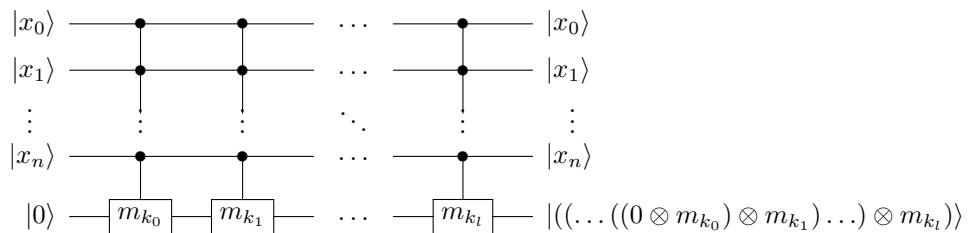
$$f(a_0, a_1, \dots, a_n) = m_{k_0} \otimes m_{k_1} \otimes \dots \otimes m_{k_l}$$

Wtedy każdy minterm można obliczyć za pomocą maksymalnie $n * 2$ bramek NOT oraz uogólnionej bramki Toffoliego rozmiaru $n + 1$. Pierwsze maksymalnie n bramek NOT jest używanych do otrzymania wartości literalów tworzących minterm. Następnie bramka Toffoliego jest wykorzystana do obliczenia wartościowania mintermu i zapisania go na bicie pomocniczym. Pozostałe bramki NOT są wykorzystane do przywrócenia stanów wejściowych na bitach argumentów.

Dla przykładowego mintermu $m = x_0 x_1 \overline{x_2} x_3 x_4$ konstrukcja ta wygląda następująco:



Korzystając z tak zbudowanych fragmentów układu obliczających wartości mintermów można stworzyć następujący układ kwantowy.



Ostatni bit powyższego układu zbiera wartości mintermów. Jeśli na wejściu był on zerem to na wyjściu będzie

przechowywał wartość funkcji f .

◆

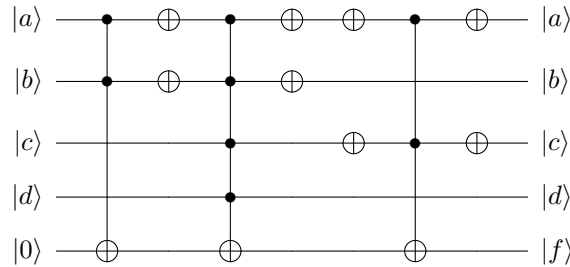
5.5.1 Przykład

Weźmy przykładową funkcję

$$f(a,b,c,d) = ab \otimes \bar{a}bcd \otimes \bar{a}\bar{c}$$

w postaci alternatywy wykluczającej termów.

Wtedy układ kwantowy, zbudowany z bramek *NOT* oraz bramek Toffoliego, wyrażający tę funkcję, wygląda następująco:



5.5.2 Algorytm

W celu optymalizacji bramek *NOT* wykorzystanych w układzie obliczającym funkcję f algorytm przechowuje informację o aktualnym stanie zmiennych. To znaczy informację czy kubit odpowiadający danej zmiennej przechowuje jej wartość czy negację tej wartości. Dzięki temu wartości argumentów nie są przywracane po każdej bramce Toffoliego, ale jedynie raz na sam koniec.

Pseudokod 5.1: Konwersja postaci ESOP to listy bramek kwantowych

Input: Zbiór termów funkcji w postaci ESOP E , index wyjścia o

Output: Lista bramek kwantowych G

```

1  negPol ← pusta lista ;                               // zbiór aktualnie zagenowanych zmiennych
2  G ← pusta lista ;
3  foreach t ∈ E do
4      foreach v ∈ t do
5          if v postaci  $\bar{x}$ , gdzie x to zmienna then
6              if v.id ∉ negPol then
7                  do G dodaj not(v.id) ;
8                  do negPol dodaj v.id ;
9              else
10                 if v.id ∈ negPol then
11                     do G dodaj not(v.id) ;
12                     z negPol usuń v.id ;
13 do G dodaj tfl(lista id elementów z t)(o) ;
/* przywrócenie stanu początkowego                                     */
14 foreach i ∈ negPol do
15     do G dodaj not(i.id) ;

```

5.6 Rozwinięcie Shannona

Wejściem do translatora są funkcje boolowskie zdefiniowane za pomocą bramek logicznych *AND*, *NOT*, *OR* oraz *XOR*. Zatem, że przekształcić każdą z tych funkcji na serię bramek kwantowych muszą być najpierw przekształcone do postaci alternatywy wykluczającej termów. Najprostszym podejściem mogłoby być wyliczenie całej tabeli funkcji i przedstawienie jej jako alternatywa wykluczająca mintermów.

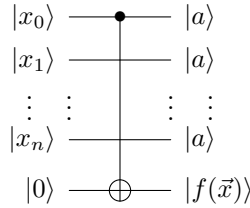


Weźmy następująco zdefiniowaną funkcję

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

gdzie \vec{x} jest wektorem długości $n \in \mathbb{N}$, a h funkcją stałą równą 1.

Do wyrażenia tej funkcji potrzeba $\frac{1}{2}n$ mintermów, które zostaną zamienione na $\frac{1}{2}n$ bramek Toffoliego. Jest to wysoko nieoptymalne, kiedy do obliczenia tej funkcji wystarczy jedna bramka Feymana.



Zamiast wyliczać całą tabelę wartościowań dla funkcji boolowskiej można zbudować drzewo decyzyjne.

Definicja 5.1 Rozwinięcie Shannona dla funkcji boolowskiej f względem zmiennej x_i definiuje się następująco

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

gdzie

$$f_{x_i} = f(x_i = 1)$$

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

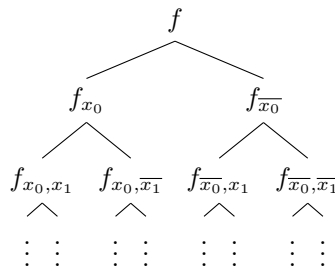
To znaczy, f_{x_i} jest funkcją powstałą przez podstawienie za x_i 1.

5.6.1 Algorytm

Korzystając z rozwinięć Shannona budujemy drzewo w następujący sposób:

1. Podstaw pod korzeń funkcję f . Niech $f' := f$.
2. Jeśli f' nie jest funkcją stałą przejdź do kolejnego kroku, w przeciwnym przypadku koniec.
3. Niech a będzie dowolnym z argumentów funkcji f' .
4. Podstaw jako prawego syna węzła z f' f'_a , jako lewego $f'_{\overline{a}}$.
5. Wykonaj kroki od 2. dla synów f' .

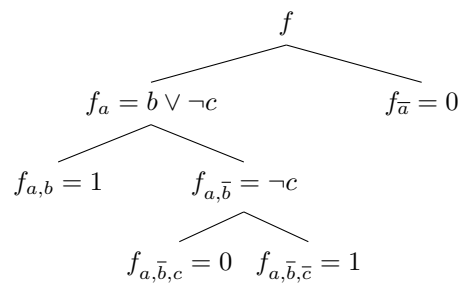
Algorytm ten zwraca drzewo kolejnych rozwinięć o następującej strukturze:



Korzystając z tak zbudowanego drzewa można łatwo znaleźć postać alternatywy wykluczającej termów funkcji f . Każdy liść zawiera funkcję stałą, której odpowiada term. Na przykład dla liścia $f_{x_0, \overline{x_1}, x_2}$ mamy term $x_0 \overline{x_1} x_2$. Wartość tej funkcji jest wskaźnikiem czy dany term należy do funkcji f .

Przykład

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



Stąd

$$f(a,b,c) = ab \oplus a\bar{b}\bar{c}$$



Implementacja systemu

6.1 Opis technologii

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

6.2 Instrukcja obsługi

Warunkiem wstępnym do skorzystania z programu jest posiadanie zainstalowanego Java JDK w wersji 8 lub wyższej. Program przyjmuje na wejściu jeden lub dwa argumenty. Pierwszym jest ścieżka do pliku zawierającego instrukcje napisane przy użyciu języka opisanego w rozdziale 4. Drugim argumentem jest ścieżka do pliku, do którego ma zostać zapisane rozwiązanie. W przypadku braku drugiego argumentu rozwiązanie zostanie wypisane bezpośrednio na konsolę.

6.3 Język wyjścia

a	<1>	-H-	---	-●-	-x-	-o-	-x-	<0>	0.0
b	<0>	---	---	---	---	---	---	<0>	0.0
c	<0>	---	---	-●-	-●-	---	-x-	<1>	0.5
d	<1>	---	-H-	-o-	-x-	---	---	<0>	0.5

W pierwszej kolumnie widać nazwy zmiennych, które odpowiadają kubitom. Następna kolumna zawiera wartości początkowe.

Dalej jest widoczny układ kwantowy. 'H' to bramka Hadamarda, bramka Toffoliego oznaczona jest przez 'o' dla bitu wejściowego i '●' dla bitów sterujących, w szczególności w piątej kolumnie powyższego układu widać bramkę NOT. Bramka Fredkina analogicznie wykorzystuje oznaczenie '●' dla bitów sterujących oraz, tak samo jak bramka SWAP, 'x' dla bitów wejściowych.

Po schemacie układu kwantowego są wartości wynikowe dla każdego z kubitów, czyli stany kubitów po zmierzeniu ich wartości oraz na końcu prawdopodobieństwo zmierzenia stanu $|1\rangle$ dla każdego z kubitów w rejestrze.

¹<https://github.com/scalanlp/breeze>

²<https://github.com/scala/scala-parser-combinators>



Przykłady użycia

7.1 Funkcja boolowska

7.1.1 Wejście

```
a = 1
b = 0
c = 0
f = and(and(or(a, b), a), not(or(b, c)))
```

7.1.2 Wyjście

```
a <1> --- --- -●- --- --- <1> 1.0
b <0> -○- --- -●- --- -○- <0> 0.0
c <0> --- -○- -●- -○- --- <0> 0.0
f <0> --- --- -○- --- --- <1> 1.0
```

7.2 Układ logiczny z wartościami 1/2

7.2.1 Wejście

```
a = 0
b = 0
c = 0
d = 1
hdm(a)
hdm(b)
hdm(c)
f_one = and(a, b, c)
f_two = and(a, d, or(a, c))
```

7.2.2 Wyjście

```
a <0> -H- --- --- -●- -●- <0> 0.5
b <0> --- -H- --- -●- --- <0> 0.5
c <0> --- --- -H- -●- --- <1> 0.5
d <1> --- --- --- --- -●- <1> 1.0
f_one <0> --- --- --- -○- --- <0> 0.125
f_two <0> --- --- --- --- -○- <0> 0.5
```



7.3 Bramki kwantowe

7.3.1 Wejście

```

a = 1
b = 0
c = 1
d = 1
hdm(b)
tfl(:b, c)
hdm(c)
frd(:c, b, a)
tfl(:a, d)
hdm(a)

```

7.3.2 Wyjście

```

a <1> --- --- --- -x- -●- -H- <0> 0.5
b <0> -H- -●- --- -x- --- --- <1> 0.75
c <1> --- -○- -H- -●- --- --- <1> 0.5
d <1> --- --- --- --- -○- --- <1> 0.25

```

Podsumowanie

W pracy udało się spełnić wszystkie pierwotne wymagania. W jej wyniku został stworzony program, który pozwala na zamodelowanie na wejściu układu logicznego i tworzy odpowiadający mu układ kwantowy. Program symuluje również działanie układu kwantowego dla pewnego podzbioru bramek kwantowych.

8.1 Minimalizacja liczby kubitów pomocniczych

Głównym zainteresowaniem optymalizacyjnym w tej pracy było generowanie układów kwantowych w sposób, który minimalizuje używane kubity pomocnicze. Program w tym celu wykorzystuje algorytm oparty o używanie postaci ESOP dla funkcji boolowskich, a następnie zamianę tej postaci na ciąg bramek z rodziny Toffoliego oraz bramek *NOT*. W wyniku zastosowania tej metody wykorzystywany jest jedynie jeden kubit pomocniczy dla każdej wyjściowej funkcji boolowskiej. Na wyjściu ten kubit przechowuje wartość tej funkcji.

8.1.1 Wnioski i uwagi

Podczas przeprowadzania obliczeń kwantowych bardzo istotnym czynnikiem jest czas działania. Zjawisko kwantowej dekoherencji powoduje, że czym dłużej obliczenia trwają tym mniej dokładne zwracają wyniki.

Optymalizacja postaci ESOP

Zwracana przez kolejne rozwinięcia SHannona postać ESOP często nie jest najbardziej optymalną. Istnieją heurystyki, które służą optymalizacji postaci ESOP funkcji boolowskich (np. Exorcism-MV-2). Głównym celem tych optymalizacji jest minimalizacja liczby termów wykorzystywanych do zapisu funkcji.

Rodzina bramek Toffoliego

Z każdą bramką kwantową związany jest pewien koszt. Koszt ten jest zależny między innymi od wielkości bramki. Układy kwantowe pozwalają z reguły na wykorzystywanie tylko niewielkiego zbioru bramek kwantowych ograniczając się to tych, które operują na maksymalnie dwóch kubitach. W wyniku tego wielokubitowe bramki Toffoliego muszą zostać rozbite do ciągu mniejszych bramek. Możliwe jest rozbicie n -kubitowej bramki Toffoliego na maksymalnie dwu-kubitowe nie dodając bitów pomocniczych. Prowadzi to jednak do znacznego

źródło

Kubity pomocnicze

Największe istniejące komputery kwantowe nie przekraczają rozmiaru 53 kubitów, zatem łatwo widać, że kubity są cennym zasobem podczas obliczeń i warto minimalizować ich użycie. Żeby jednak komputery kwantowe mogły pracować i wykonywać realne obliczenia liczba kubitów musi znacznie wzrosnąć. Stan pierwotny kubitów pomocniczych, które przechowują niepotrzebne wyniki, może zostać przywrócony dzięki odwołaniu obliczeń kwantowych. Zatem te same kubity mogą być używane wielokrotnie.



Bibliografia

- [1] G. Bacciagaluppi. The role of decoherence in quantum mechanics. E. N. Zalta, redaktor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, wydanie fall 2016, 2016.
- [2] K. Fazel, M. Thornton, J. Rice. Esop-based toffoli gate cascade generation. strony 206 – 209, 09 2007.
- [3] J. Hui. Quantum computing series. Strona: <https://medium.com/@jonathan.hui/qc-quantum-computing-series-10ddd7977abd>, 2017.
- [4] A. Ligeża. 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>.
- [5] A. Muthukrishnan. Classical and quantum logic gates: An introduction to quantum computing. 1999.
- [6] M. A. Nielsen, I. L. Chuang. *Quantum Computation and Quantum Information, 10th Anniversary Edition*. Cambridge University Press, 2010.
- [7] M. Perkowski, S. Hossain. *Quantum Robotics, rozdział 6*. wydanie August 7, 2010.
- [8] M. Saeedi, M. Pedram. Linear-depth quantum circuits for n-qubit toffoli gates with no ancilla. *Physical Review A*, 87(6), Jun 2013.
- [9] 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.
- [10] D. Voudouris, M. Sampson, G. Papakonstantinou. 1variable reordering for reversible wave cascades.

