1 Bramka do uniwersalnych obliczeń

W świecie klasycznym zwykle patrzymy na obliczenia jak na "pudełko" z wejściem i wyjściem:

$$x \;—\boxed{y = f(x)}—\quad y \tag{1}$$

W świecie kwantowym bezpośrednie zastosowanie takiego "pudełka" nie jest możliwe, ponieważ każda bramka obliczeniowa musi być unitarna, a co za tym idzie, odwracalna (ponieważ dla każdej bramki U istnieje $U^{\dagger}$ będące jej odwrotnością)

Uniwersalna bramka kwantowa obliczająca funkcję $f(x)$ spełniająca warunek unitarności jest zdefiniowana jako:

$$\begin{array}{lcl} |x\rangle_n & \multirow{2}{*}{$\boxed{U_f}$} & |x\rangle_n \\ |y\rangle_m & & |y \oplus f(x)\rangle_m \end{array} \tag{2}$$

gdzie:

- $|x\rangle_n$ rejestr wejściowy o n qbitach
- $|y\rangle_m$ rejestr wyjściowy o m qbitach
- $\oplus$ oznacza funkcje xor wg tabelki 1

a	b	$a \oplus b$
0	0	0
1	0	1
0	1	1
1	1	0

Tabela 1: Tabela logiczna funkcji xor

Uwaga: rejestry wejściowy i wyjściowy "ciagną" się przez całe obliczenie i nie są intuicyjne wejściem i wyjściem z "pudełka" bramki.

2 Problem Deutscha

W podstawowym problemie Deutscha mamy 1-qbitowe wejście 1-qbitowe wyjście oraz cztery możliwe funcje zdefiniowane dla klasycznych bitów tak jak w tabelce 2.
Pytaniem jakie zadał Deutch jest: **Mamy daną jedną z czterech funkcji z tabelki 2, ale nie wiemy którą. Chcemy sprawdzić, czy jest ona stała czy zmienna. Ile razy musimy uruchomić tę funkcję, żeby to sprawdzić?**

nazwa funkcji	f(0)	f(1)	rodzaj
f_0	0	0	stała
f_1	0	1	zmienna
f_2	1	0	zmienna
f_3	1	1	stała

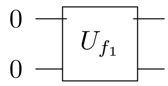
Tabela 2: Funkcje w podstawowym problemie Deutcha

2.1 Rozwiązanie klasyczne

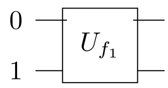
Oczywiście, musimy policzyc $f(0)$ potem $f(1)$ i porównać je. Jeśli wyniki są równe funkcja jest stała, jeśli różne - zmienna. Odpowiedź: dwa razy

2.2 Rozwiązanie kwantowe

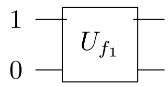
Najpierw zobaczmy jak wygląda bramka kwantowa obliczająca funkcję dowolną funkcję z tabelki 2 Bramkę tę znajdziemy analizując wszystkie możliwości jej wejść i wyjść na przykładzie f_1.



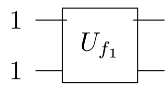
$$\begin{array}{l} 0 \\ 0 \oplus f_1(0) = 0 \oplus 0 = 0 \end{array} \tag{3}$$



$$\begin{array}{l} 0 \\ 1 \oplus f_1(0) = 1 \oplus 0 = 1 \end{array} \tag{4}$$

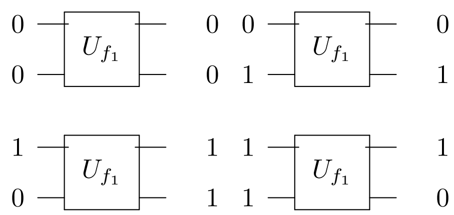


$$\begin{array}{l} 1 \\ 0 \oplus f_1(1) = 0 \oplus 1 = 1 \end{array} \tag{5}$$



$$\begin{array}{l} 1 \\ 1 \oplus f_1(1) = 1 \oplus 1 = 0 \end{array} \tag{6}$$

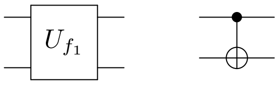
Mamy więc zdefiniowane działania bramki dla wszystkich stanów bazowych $|00\rangle$,$|01\rangle$,$|10\rangle$,$|11\rangle$

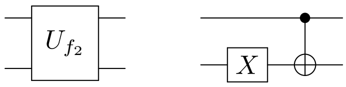


Jaka bramka realizuje to działanie ?

Bramka która realizuje takie działanie to CNOT !
Podobnie można znaleźć bramki kwantowe do pozostałych funkcji. Umieszczono je poniżej.

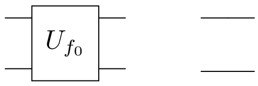
Dla grupy funkcji zmiennych mamy (po lewej symbol, po prawej realizacja):

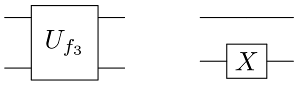




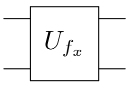
Jak widać "zmienność" czyli wpływ x na y jest realizowany poprzez CNOT.

A dla grupy funkcji stałych mamy:





Teraz pytanie Deutcha można sformuować tak: dostaliśmy bramkę:

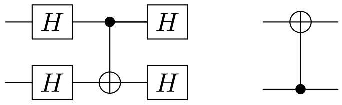


Ile razy musimy ją uruchomić, aby sprawdzić do której grupy ona należy ?

Porównując obie grupy widać wyraźnie różnicę - grupa "zmienna" zawiera bramkę CNOT. Jak ją wykryć bez patrzenia do środka uruchamiając tę bramkę minimalną ilość razy ?

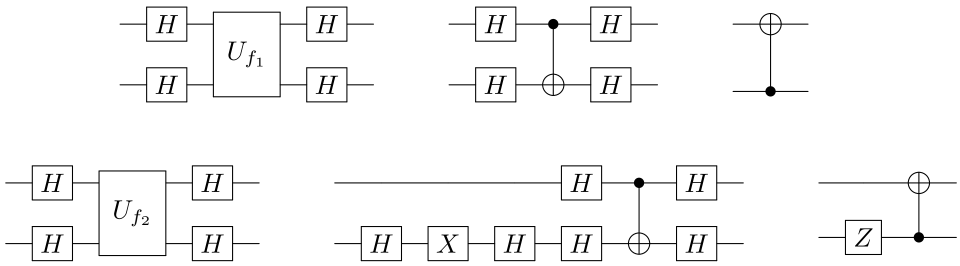
Nie możemy po prostu dać stanu $|1\rangle$ na górnym qbicie i sprawdzić, czy ta bramka tam jest, ponieważ wtedy nie rozróżnilibyśmy bramek U_{f1} i U_{f3} oraz U_{f2} i U_{f0} - zachowywałyby się one tak samo (i to niezależnie od tego, co dalibyśmy na dolnym qbicie).

Rozwiązaniem kwantowym jest "odwrócenie do góry nogami" bramki CNOT i "zneutralizowanie" działania bramki X poprzez skorzystanie z tożsamości:

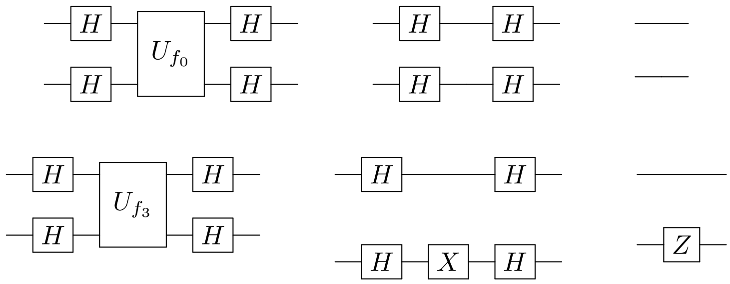


Oraz tożsamości: HXH=Z i HH=I

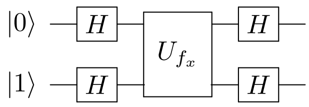
Po obłożeniu bramki U_f bramkami H z każdej strony otrzymujemy:
Grupa funkcji zmiennych:



Grupa funkcji stałych:



Teraz łatwo widać, że ustawiając DOLNY qbit na $|1\rangle$ możemy rozróżnić grupę funkcji zmiennych od stałych poprzez wykrycie CNOTa uruchamiając każdą bramkę tylko JEDEN raz.



$zmienna : |1\rangle\, , stala\, |0\rangle$

Uwaga:

- Uzyskaliśmy przyspieszenie na komputerze kwantowym - zamiast liczyć coś dwa razy, liczymy tylko raz
- cena jaką płacimy: po tej operacji nie znamy numeru funkcji, a jedynie do jakiej grupy należy
- wynik pojawił się na rejestrze wejściowym (charakterystyczne dla algorytmów kwantowych)
- W Merminie w rozdziale 2 można znaleźć matematyczne wyliczenie tego problemu
- W Merminie na górnym qbicie jest ustawiany $|1\rangle$ - nie ma to znaczenia, dopóki wiemy jak odczytać wynik (i gdzie)

a=	0	1	1	0	1
x=	1	0	1	0	1
a$\cdot x$	0	0	1	0	1
f(x, a)	$0 \oplus 0 \oplus 1 \oplus 0 \oplus 1$				
f(x, a)	=0				

Tabela 3: Przykład działania f(x,a) w problemie Bernsteina-Vezziraniego

a=	0	1	1	0	1
x=	1	0	0	0	0
a$\cdot x$	0	0	0	0	0
f(x, a)	$0 \oplus 0 \oplus 0 \oplus 0 \oplus 0$				
f(x, a)	=0				

Tabela 4: Przykład obliczania najstarszego bitu parametru a w problemie Bernsteina-Vaziraniego

3 Problem Bersteina-Vaziraniego

Funkcja z problemu Bersteina-Vaziraniego jest zdefiniowana przy założonym n-bitowym parametrze a, dla n-qbitowego rejestru wejściowego oraz 1-qbitowego rejestru. Przykład jej działania pokazuje tabelka 3 (przykład wyjaśniony na filmiku).

Pytanie zadane w problemie brzmi: **mamy daną funkcję f, ale nieznamy jej (ustalonego z góry) parametru a, ile razy musimy uruchomić tę funkcję, żeby dowiedzieć się, ile wynosi a?**

3.1 Rozwiązanie klasyczne

Klasycznie musielibyśmy uruchamiać tę funkcję n razy dla wartości x, które w swojej bitowej postaci mają tylko jedną jedynkę. Dla przykładu z tabelki 3 byłyby to $x = 10000$, $x = 01000$, $x = 00100$, $x = 00010$, $x = 00001$. Wtedy x działa jak taka "maska" wyłuskująca wartość pojedynczego bitu. Odpowiedź na pytanie zadane w problemie: n razy

a=	0	1	1	0	1
x=	0	1	0	0	0
a$\cdot x$	0	1	0	0	0
f(x, a)	$0 \oplus 1 \oplus 0 \oplus 0 \oplus 0$				
f(x, a)	=1				

Tabela 5: Przykład obliczania drugiego w kolejności bitu parametru a w problemie Bernsteina-Vaziraniego

3.2 Rozwiązanie kwantowe

Najpierw zbudujmy bramkę U_f obliczającą naszą funkcję $f(x,a)$. Jest ona pokazana na rys 2.8 w rozdziale [http://www.lassp.cornell.edu/mermin/qcomp/chap2.pdf] Można łatwo zweryfikować, że działa poprawnie.

Następnie zastosujemy metodę "obkładania Hadamardami" stosowaną już w problemie Deutcha. Wynik jest pokazany na rys 2.9 w/w rozdziału. Hadamardy "odwracają do góry nogami" wszyskie CNOTy. Wtedy ustawiając $|1\rangle$ na rejestrze wyjściowym włączamy wszystkie bramki CNOT "na raz". Jeśli ustawiliśmy $|0\rangle$ na qbitach rejestru wejściowego, łatwo możemy wykryć, gdzie znajdowały się bity kontrolne, a co za tym idzie, które bity a mają wartość 1.

Wniosek: Można znaleźć wartość a uruchamiając funkcję obliczającą f tylko JEDEN raz.

- Otrzymaliśmy przyspieszenie - zamiast obliczać funkcję n-razy obliczamy ją tylko raz.
- wynik znowu pojawił się na rejestrze wejściowym (!).