日本語訳『Qiskit Textbook』勉強会 第2章 2.3, 2.4, 2.5

Kifumi Numata

2.3 位相キックバック

1. CNOTゲートの探索
2. 位相キックバック
   2.1 CNOT回路の等価性の説明
   2.2 Tゲートでのキックバック

2.4 普遍性の証明

1. はじめに
2. 普遍性の定義
3. 普遍性の証明
4. 普遍的量子ゲートのセット

2.5 基本的な回路の等価性

1. 制御ZをCNOTから作成する
2. 量子ビットのスワップ
3. 制御回転
4. トフォリゲート
5. HとTによる任意の回転

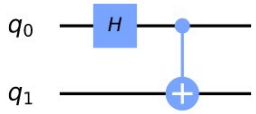
2.3 位相キックバック

目次

2.3-1 CNOTゲートの探索

ビット並び：|q1 q0⟩です。
CNOT：制御がq0、標的がq1です。

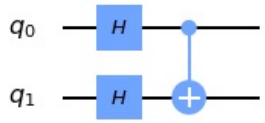
|0+>にCNOT：エンタングルメントをつくる



$$\mathrm{CNOT}|0+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

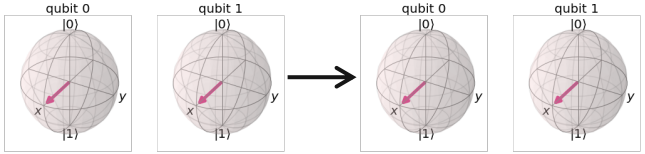
2.2章に出てきたCNOT：エンタングルメントを作る

|++>にCNOT：変わらない



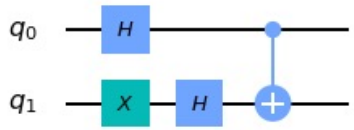
$$|++\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

$$\mathrm{CNOT}|++\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$



CNOTは、|01⟩と|11⟩の振幅をスワップする。
|++⟩ は、CNOTを作用しても状態が変わらない

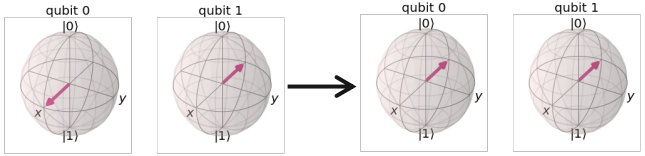
|- +>にCNOT：制御ビット側が – に



$$|-+\rangle = \frac{1}{2}(|00\rangle + |01\rangle - |10\rangle - |11\rangle)$$

$$\mathrm{CNOT}|-+\rangle = \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle + |11\rangle)$$

$$\mathrm{CNOT}|-+\rangle = |--\rangle$$

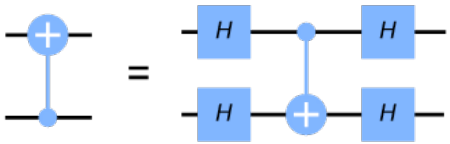


|01⟩ と |11⟩ の振幅がスワップされる。
|- +⟩へのCNOTは、*標的*量子ビットの状態を変更せずに、
*制御*量子ビットの状態に影響を与える。

2.3-1 CNOTゲートの探索 つづき

ビット並び：|q1 q0⟩です。
CNOT_{01} ：制御q0から標的q1へのCNOT
CNOT_{10} ：制御q1から標的q0へのCNOT

CNOTをHゲートで挟む：制御と標的が逆になる



なぜなら、
CNOTと|++⟩,|- +⟩, |+ -⟩, |- -⟩の関係は…

$$|{+}{+}\rangle \rightarrow \mathrm{CNOT}_{01} \rightarrow |{+}{+}\rangle$$
$$|{-}{+}\rangle \rightarrow \mathrm{CNOT}_{01} \rightarrow |{-}{-}\rangle$$
$$|{+}{-}\rangle \rightarrow \mathrm{CNOT}_{01} \rightarrow |{+}{-}\rangle$$
$$|{-}{-}\rangle \rightarrow \mathrm{CNOT}_{01} \rightarrow |{-}{+}\rangle$$

$$|00\rangle \rightarrow (HH) \rightarrow |{+}{+}\rangle \rightarrow \mathrm{CNOT}_{01} \rightarrow |{+}{+}\rangle \rightarrow (HH) \rightarrow |00\rangle$$
$$|10\rangle \rightarrow (HH) \rightarrow |{-}{+}\rangle \rightarrow \mathrm{CNOT}_{01} \rightarrow |{-}{-}\rangle \rightarrow (HH) \rightarrow |11\rangle$$
$$|01\rangle \rightarrow (HH) \rightarrow |{+}{-}\rangle \rightarrow \mathrm{CNOT}_{01} \rightarrow |{+}{-}\rangle \rightarrow (HH) \rightarrow |01\rangle$$
$$|11\rangle \rightarrow (HH) \rightarrow |{-}{-}\rangle \rightarrow \mathrm{CNOT}_{01} \rightarrow |{-}{+}\rangle \rightarrow (HH) \rightarrow |10\rangle$$

$$\| $$
$$\mathrm{CNOT}_{10}$$
に等しい

Qiskit

Qiskitのユニタリーシミュレーターで行列を確かめても同じです。

2.3-2 位相キックバック

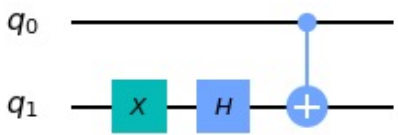
2.1 CNOT回路の等価性の説明

位相キックバックは、ほとんどすべての量子アルゴリズムで使用されているほど重要です。

キックバックとは、ゲートによって量子ビットに追加された固有値（位相）が、
制御操作によって制御ビットに「キックバック」されることです。

例えば、$|-\rangle$ の量子ビットにXゲートを実行すると、位相 -1が得られます：$X|-\rangle = -|-\rangle$ これを使った例が次。

位相キックバックの例 ：　標的が|->で、制御が$|1\rangle$のとき、$X|-\rangle$の位相（-1）が状態全体に影響する例
（グローバル位相であるため、観測はされません。）

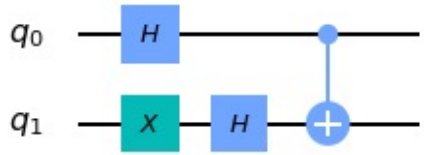


$$\begin{aligned}\mathrm{CNOT}|-0\rangle &= |-\rangle \otimes |0\rangle \\ &= |-0\rangle\end{aligned}$$

$$\begin{aligned}\mathrm{CNOT}|-1\rangle &= X|-\rangle \otimes |1\rangle \qquad (\ X|-\rangle = -|-\rangle \ \text{より}) \\ &= -|-\rangle \otimes |1\rangle \\ &= -|-1\rangle\end{aligned}$$

位相キックバックの場合：制御量子ビット側に相対位相が追加される例

先ほどの回路の制御側を重ね合わせに

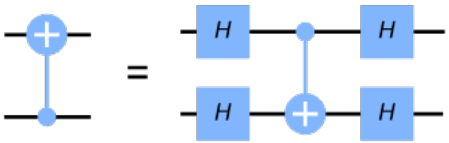


$$
\begin{aligned}
\mathrm{CNOT}|{-}{+}\rangle &= \tfrac{1}{\sqrt{2}}(\mathrm{CNOT}|{-}0\rangle + \mathrm{CNOT}|{-}1\rangle) \\
&= \tfrac{1}{\sqrt{2}}(|{-}0\rangle + X|{-}1\rangle) \\
&= \tfrac{1}{\sqrt{2}}(|{-}0\rangle - |{-}1\rangle) \\
&= |{-}\rangle \otimes \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\
&= |{-}{-}\rangle
\end{aligned}
$$

制御量子ビットが|1⟩の場合のみ、
グローバル位相（今回はマイナス）が
標的量子ビットに適用されます。

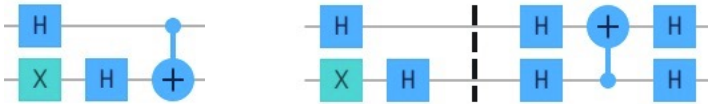
制御量子ビット側に相対位相が追加されることがわかります。

相対位相とは・・



より、上記の例で、HゲートでCNOTをラップ&制御と標的を逆にすると、
量子ビットが計算基底から(|+⟩,|−⟩) 基底に変換され、同じ効果を確認できます。

$$CNOT_{01}|-+\rangle = (H\otimes H)CNOT_{10}(H\otimes H)|-+\rangle = (H\otimes H)CNOT_{10}|10\rangle = (H\otimes H)|11\rangle = |--\rangle$$

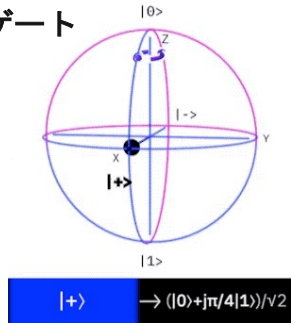


一部のハードウェアは一方向のCNOTしか実装できないので、この等価性を使って双方向のCNOTを実現します。

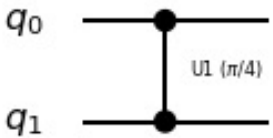
2.3-2 位相キックバック

2.2 Tゲートでのキックバック

Tゲート



$$\mathrm{T} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$$



$$\text{Controlled-T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\pi/4} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}}(1+i) \end{bmatrix}$$

QiskitでコントロールTは、
cu1(pi/4)を使います。

のちほどQiskitでやってみます

一般に

$$\mathrm{U} = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix}$$

一般の教科書配列

$$\text{Controlled-U} = \begin{bmatrix} I & 0 \\ 0 & U \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix}$$

Qiskit配列

$$\text{Controlled-U} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & u_{00} & 0 & u_{01} \\ 0 & 0 & 1 & 0 \\ 0 & u_{10} & 0 & u_{11} \end{bmatrix}$$

コントロールTは、
どっちの向きでも同じ行列。

Tゲートでのキックバック

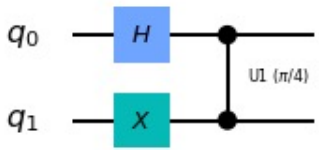
制御量子ビット側に相対位相が追加される例

$$\mathrm{T} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$$

状態 |1⟩ の量子ビットにTゲートを適用すると、その量子ビットに e^$i\pi/4$ の位相が追加されます：

$$T|1\rangle = e^{i\pi/4}|1\rangle$$ （グローバル位相で観測できません）

制御を|+⟩で、標的を|1>で制御Tゲートを作用すると、制御ビットの相対位相が変化します：



$$|1+\rangle = |1\rangle \otimes \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$= \tfrac{1}{\sqrt{2}}(|10\rangle + |11\rangle)$$

$$\text{Controlled-T}|1+\rangle = \tfrac{1}{\sqrt{2}}(|10\rangle + e^{i\pi/4}|11\rangle)$$

$$\text{Controlled-T}|1+\rangle = |1\rangle \otimes \tfrac{1}{\sqrt{2}}(|0\rangle + e^{i\pi/4}|1\rangle)$$

これは、標的量子ビットを変化させずに、制御量子ビットをZ軸回りに回転させる作用があります。

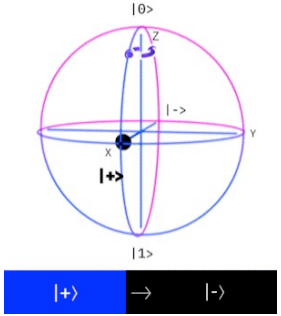
QiskitでBloch球で確認

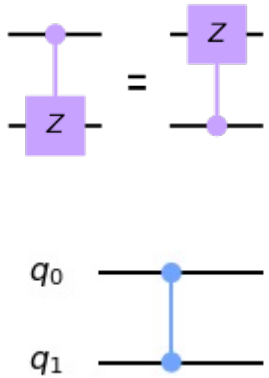
制御Zゲートもキックバックに使われます

制御Zゲートには、制御ビットと標的ビットの区別がないので、
Qiskitでは、制御Z回転ゲートを対称的な記号で表記します。

Z |1> = - |1>

Z回転ゲート





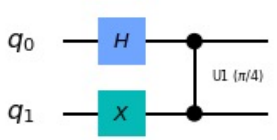
入力	CZ_01 の出力	CZ_10 の出力
00	00	00
01	01	01
10	10	10
11	-11	-11

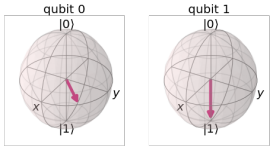
$$\text{Controlled-Z} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

位相キックバックは、制御側に1が含まれていないと作動しないが、
制御Zの場合、制御・標的の区別がないので、どちらを制御としてもよい。

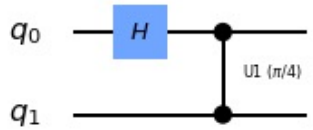
2.3 練習問題

（制御Tゲートで制御を|+⟩で、標的を|1>にすると
制御ビットの相対位相がT回転した）

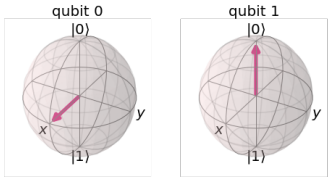




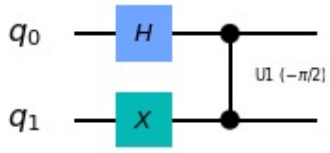
(1) 標的量子ビット（q1）が|0⟩の状態にある場合、制御量子ビット（q0）の状態はどのような結果になりますか？
Qiskitで回路を作成して、答えを確認してみましょう。



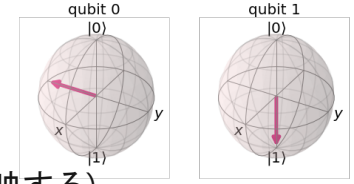
(1) 答え：|q0>=|+>のまま変わらない。



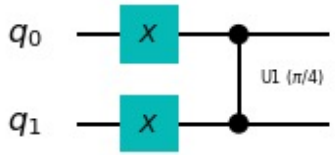
(2) 標的量子ビット（q1）が|1⟩の状態で、回路が制御Tではなく制御Sdgゲートを使用した場合、制御量子ビット（q0）はどうなりますか？



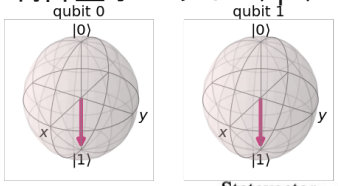
(2) 答え：|q0> = |-i>になる。(制御側にSdgの位相が反映する)



(3) 制御量子ビット（q0）の制御Tを適用する前の状態が|+⟩ではなく、|1⟩の状態にあった場合、制御量子ビット（q0）はどうなりますか？



(3) 答え：|q0> = |1>で、グルーバル位相がT回転する。



Statevector = $\begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}}(1+i) \end{bmatrix}$

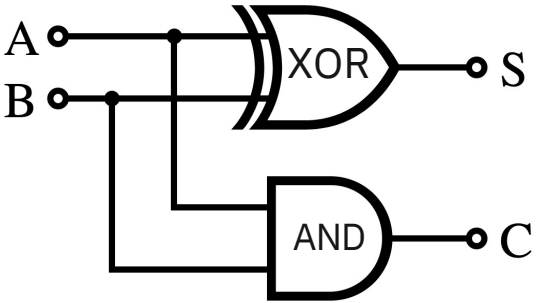
2.4 普遍性の証明

目次

2.4-1 はじめに

ある装置が、任意の入力セットを任意の出力セットへ変換することができる場合、
このことを **普遍的である**、と呼びます。

古典コンピューターでは、
1.2章の「計算の原子」で出てきた加算回路で使うゲート、AND、OR、NOTであらゆる計算の実装が可能です。
つまり、**AND、OR、NOTのセットが古典の普遍ゲート**です。（万能計算のためのゲート）



古典コンピューターは、このゲートセットで理論的にすべてを行うことができますが、一部のタスクではリソースがかかりすぎます。

例）加算回路では、数字の桁数を2倍にすると、小規模な加算回路の数が2倍必要になります。

その他、入力サイズに応じて必要なリソースが指数関数的に大きくなる問題もあります。（例： 因数分解）

量子コンピューターは、根本的に異なる方法で、この普遍性を実現することにより、この問題を軽減します。

2.4-2 普遍性の定義

デジタルコンピューターの普遍性：
「任意のビット列の組み合わせの入力」を「任意のビット列の組み合わせの出力」
に変換するゲートのセットで実現。

量子コンピューターの普遍性：
「任意の直交した入力状態の組み合わせ」を「任意の直交した出力状態の組み合わせ」
に変換するユニタリー演算のセットで実現。

入力状態・出力状態とは、
・量子形式で表現されたビット列。
または
・実際の物理システム。

$$|\psi(t)\rangle = \hat{U}(t)\,|\psi(t_0)\rangle \quad , U = e^{i\gamma H}$$

このときユニタリー演算は時間発展に相当。
ユニタリー演算をエルミート行列の指数形式で表現すると、その行列はハミルトニアンに相当。

→ 任意のユニタリー演算の実行は、任意の時間発展をシミュレーションし、任意のハミルトニアンの効果を設計することに相当。（古典にはない量子コンピューターの自然な応用例）

つまり、量子コンピューターの普遍性とは、任意の数の量子ビットで任意のユニタリー演算を実現する機能です。

2.4-3 普遍性の証明

やりたいこと：量子コンピューターで、いくつかの基本ゲートを使用して普遍性を実現できることの証明。

まず１量子ビットに対して
以下のように任意のユニタリー演算を実装するとします。

$$U = e^{i(aX+bZ)}$$

ここでゲートは、 $R_x(\theta) = e^{i\frac{\theta}{2}X}$ $R_z(\theta) = e^{i\frac{\theta}{2}Z}$ の二つです。

この任意のユニタリー演算Uは、基本のRxゲートとRzゲートから作れるのでしょうか？

XとZが可換でないので（ XZ ≠ ZX なので）、単に続けて適用するだけではUは作れません。

$$e^{iaX}e^{ibZ} \neq e^{i(aX+bZ)}$$

そこで、RxとRzをそれぞれn個に分割して、Uをこのように変形します：

$$U = \lim_{n\to\infty}\left(e^{iaX/n}e^{ibZ/n}\right)^n .$$ （Trotterの公式）

$U = e^{i(aX+bZ)}$ を $e^{i(aX+bZ)} = \lim_{n\to\infty} \left(e^{iaX/n} e^{ibZ/n}\right)^n$ に変形します（Trotterの公式）。

これは、XとZが可換でなくても成り立ち、$e^{iaX/n}$ などをテイラー展開すると導けます。

$$U = \lim_{n\to\infty} \left(e^{iaX/n} e^{ibZ/n}\right)^n .$$

U をn 個の小さなスライスに分割したものは、以下のように近似できます。（テイラー展開より）

$$e^{iaX/n} e^{ibZ/n} = e^{i(aX+bZ)/n} + O\left(\frac{1}{n^2}\right)$$

この近似の誤差は、(1/n)^2に比例。

つまり、RxゲートとRzゲートをn 個でスライスした演算 $e^{iaX/n} e^{ibZ/n}$ をn回繰り返すと、
目標とする任意のユニタリー演算の近似値が誤差1/n のオーダーで得られます。

したがって、 RxとRzをそれぞれn個に分割する分割数nを増やすだけで、必要なだけUに近づけることができます。

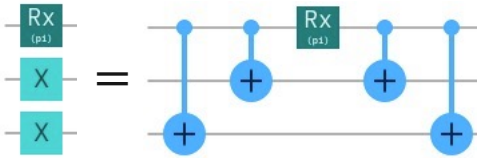
以上より、1量子ビットの演算について、X軸回転とZ軸回転があれば、任意の回転演算が作れる（十分近似できる）ことがわかりました。

複数ビットの演算の場合
以下のユニタリー演算を考えます（３量子ビット）。

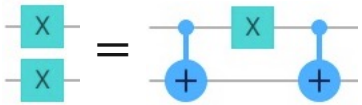
$$U = e^{i(aX\otimes X\otimes X + bZ\otimes Z\otimes Z)}.$$

単一量子ビット $R_x(\theta)$ と2つの制御NOTからユニタリー $e^{i\frac{\theta}{2}X\otimes X\otimes X}$ を作成する方法は
（今後、解説修正が入る予定）

θ=πのとき

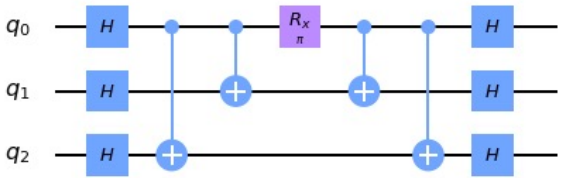


作り方：



と同様のはず

$e^{i\frac{\theta}{2}Z\otimes Z\otimes Z}$ をつくるには、Hで挟んで、できるのでは・・ （今後、解説修正が入る予定）



３量子ビットの場合もスライスを使って近似できます。

$$e^{iaX\otimes X\otimes X/n}e^{ibZ\otimes Z\otimes Z/n} = e^{i(aX\otimes X\otimes X + bZ\otimes Z\otimes Z)/n}.$$

同様にn量子ビットの場合もRx、H、CNOTゲートを使用して普遍性が十分高い精度で実現できます。

よって、Qiskitにある基本的な演算のみを使用してすべてのマルチ量子ビットユニタリー演算が再現できます。

2.4-4 普遍的量子ゲートのセット

普遍的量子ゲートにはさまざまなセットがあります。
前の章では、 Rx、H、CNOTゲートのセットを使いました。

IBMQX2プロセッサーの基本ゲートは、 U1、U2、U3、CX、Id です。
（このセットでユニタリー演算を実現する。）

他のタイプの量子コンピューターには、異なるネイティブゲートがある。
ここでは説明しません。

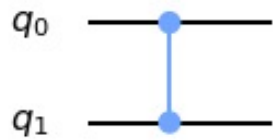
現時点で知っておく必要があるのは、
「1組の普遍的ゲートを使用して作成したアルゴリズムは、どの普遍的量子コンピューターでも実行できる」
ということです。

2.5 基本的な回路の等価性

目次

2.5-1 制御ZをCNOTから作成する

制御Z



入力		出力	
制御	標的	制御	標的
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	-1

IBM Qデバイスでは、直接適用できる2量子ビットゲートはCNOTだけです。従って、変換する方法が必要です。

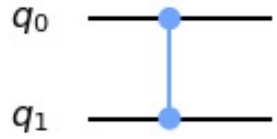
$$HXH = Z$$
$$HZH = X.$$

H|0>→|+>, H|1>→| ->
X：|0>⇆|1>
Z：|+>⇆| -> より

実際にやってみるとわかる

2.5-1 制御ZをCNOTから作成する

制御Z

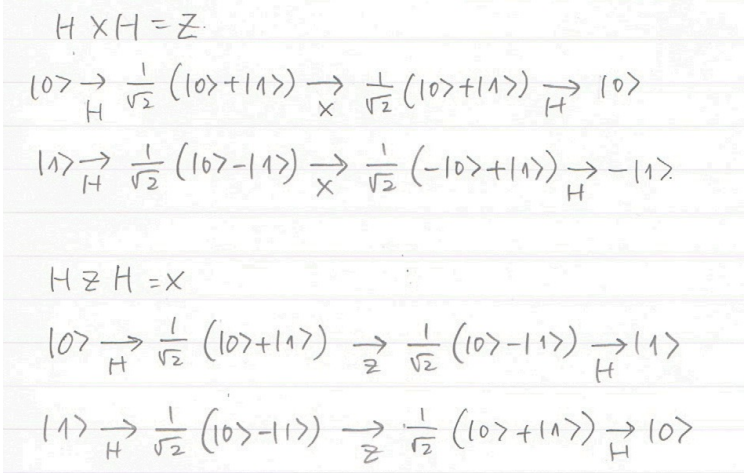


入力		出力	
制御	標的	制御	標的
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	-1

IBM Qデバイスでは、直接適用できる2量子ビットゲートはCNOTだけです。従って、変換する方法が必要です。

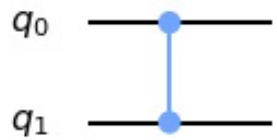
$$HXH = Z$$
$$HZH = X.$$

H|0>→|+>, H|1>→| ->
X：|0>⇆|1>
Z：|+>⇆| -> より



2.5-1 制御ZをCNOTから作成する

制御Z



入力		出力	
制御	標的	制御	標的
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	-1

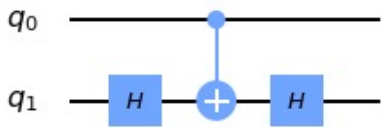
IBM Qデバイスでは、直接適用できる2量子ビットゲートはCNOTだけです。従って、変換する方法が必要です。

$$HXH = Z$$
$$HZH = X.$$

H|0>→|+>, H|1>→| ->
X：|0>⇆|1>
Z：|+>⇆| -> より

同じトリックを使用して、CNOTを制御zに変換できます。

CZ =

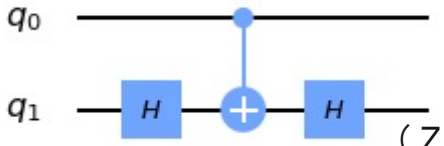


ブロッホ球で確認： https://javafxpert.github.io/grok-bloch/

制御π回転

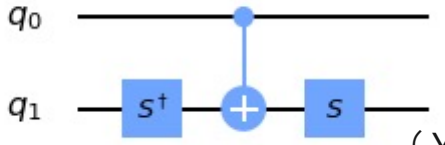
一般に、CNOTの前後に、正しい回転を配置することで、
ブロッホ球での角度πの任意の回転の制御バージョンに変換できます。

CZ =



（Z = HXHより）

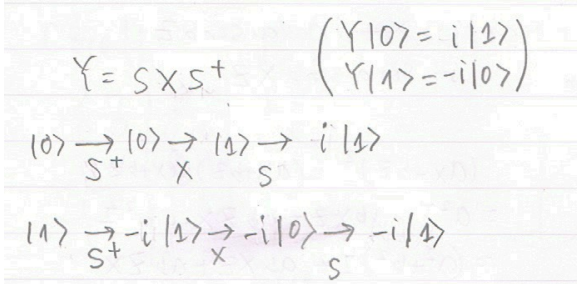
CY =

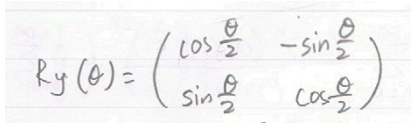


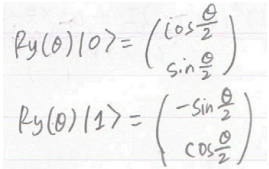
（Y = S X Sdgより）

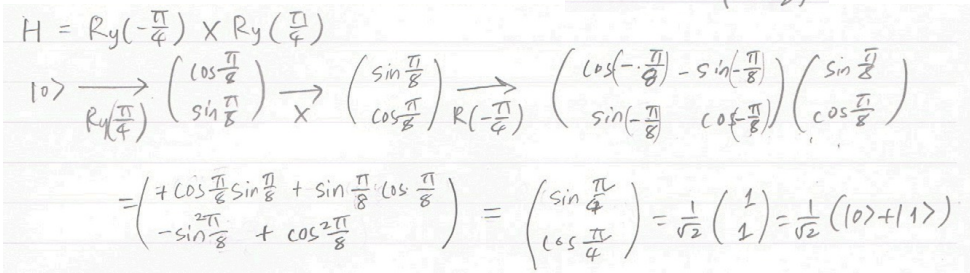
CH =

（H = Ry(-pi/4) X Ry(pi/4)より）



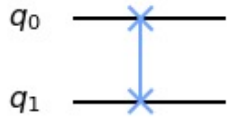




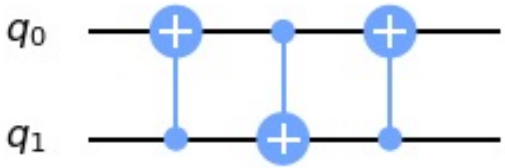


2.5-2 量子ビットのスワップ

スワップゲート

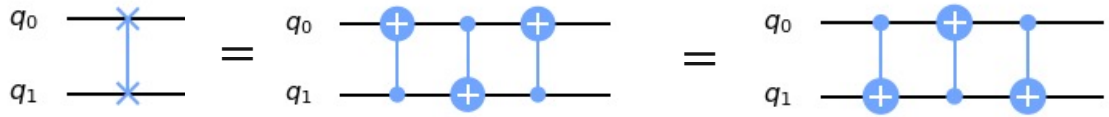


標準のゲートセットを使用してswapを作成するには…（色々書いてあるがやってみる方が早い）



入力	↑	↑	出力
00	00	00	00
01	01	**11**	**10**
10	**11**	**01**	01
11	**10**	10	**11**

q1q0の順

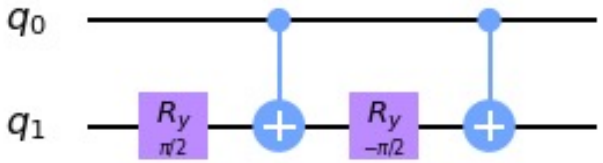


2.5-3 制御回転

（２ページ前で制御π回転をいくつか作りましたが、今回は任意の制御回転）

任意の制御回転を作ります。

制御回転 $Ry(\theta)$



制御が|0⟩ のとき：

$Ry(\theta/2)$の後に $Ry(-\theta/2)$ で変化なし。

制御が|1⟩ のとき：

$Ry(-\theta/2)$の前後にXゲートが適用されます。

これは、y回転の方向を反転するので、$Ry(\theta/2)$ と同じ。

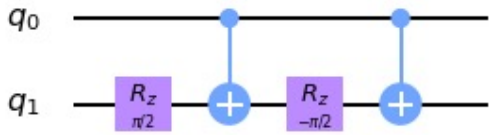
よって、$Ry(\theta/2)$が2回で$Ry(\theta)$となる。

ブロッホ球で確認：https://javafxpert.github.io/grok-bloch/

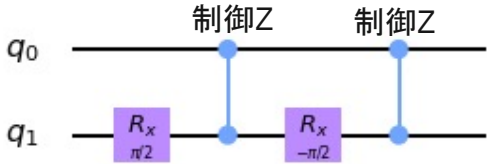
この方法が機能するのは、xとy軸が直行し、そのためにxゲートが回転の方法を反転させるため。

同じように

制御$Rz(\theta)$



制御 $Rx(\theta)$

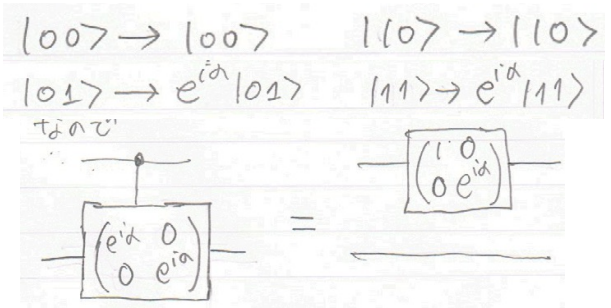


任意の制御回転を作ります。

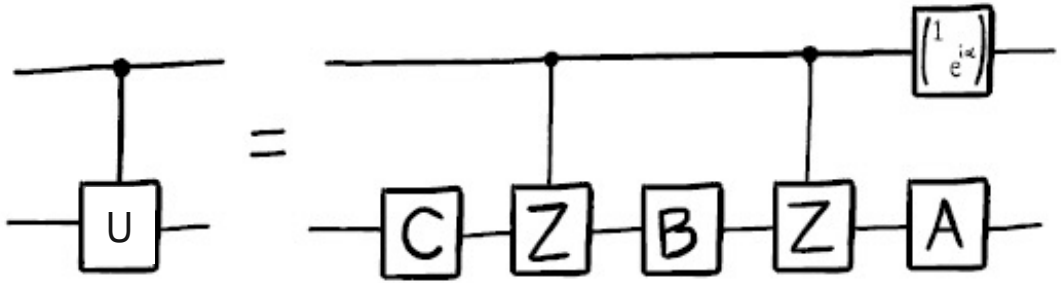
Uが単一量子ビットに働くユニタリーゲートとすると、
単一量子ビットに働くユニタリー回転A、B、Cが存在して、それは、$ABC = I, \quad e^{i\alpha} AZBZC = U$
を満たします。ここで、αは全体の位相因子。(このようにかけるUがあることは定理)

例） A=Rx(β)Ry(γ/2), B=Ry(- γ/2)Rx(-(δ+β)/2), C=Rx((δ -β)/2)
（制御Zなので、前後の回転をYとXにして角度を半分に割るようなゲートになる）

位相因子のところ（位相キックバック）
|q1q0>でq0が1のときe^iαが実行される

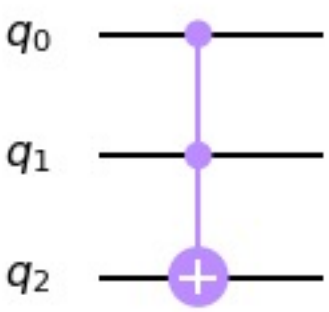


このA、B、Cを使うと、以下のように任意の制御回転が作れます。



制御が0なら標的にはABC=Iがかかり変わらない。
制御が1のとき、標的にはe^(iα) AZBZCがかかる。

2.5-4 トフォリ



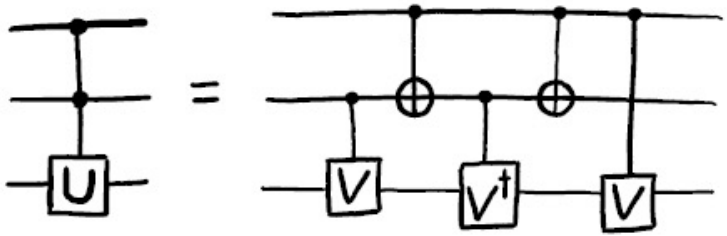
2つの制御と1つの標的を持つ３量子ビットゲートです。
両方の制御ビットが |1⟩ の状態の時のみ標的ビットにXを適用します。

トフォリは、制御・制御NOTとも考えることもでき、CCXゲートとも呼ばれます。

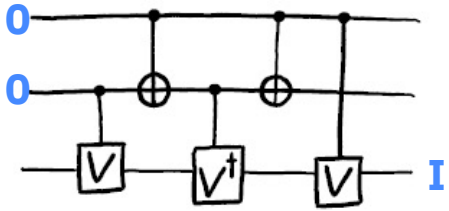
単一量子ビットゲートと2量子ビットゲートからトフォリゲートを構築します。

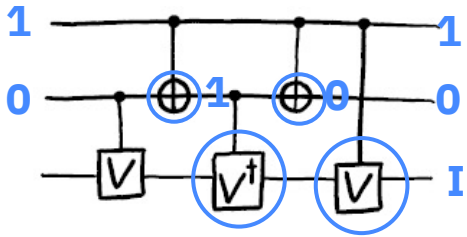
まず、

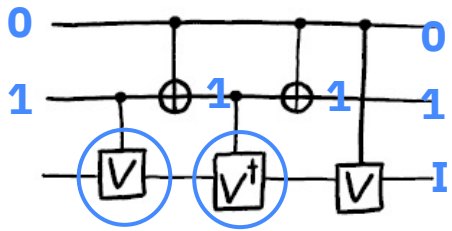
$V = \sqrt{U}$ のとき、制御Uは以下の回路に変換できます。

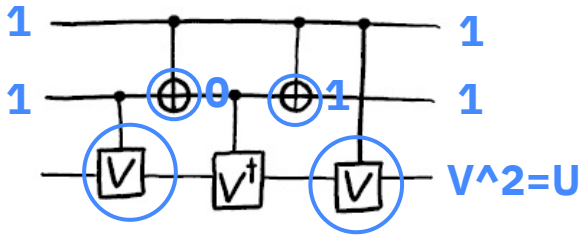


なぜなら









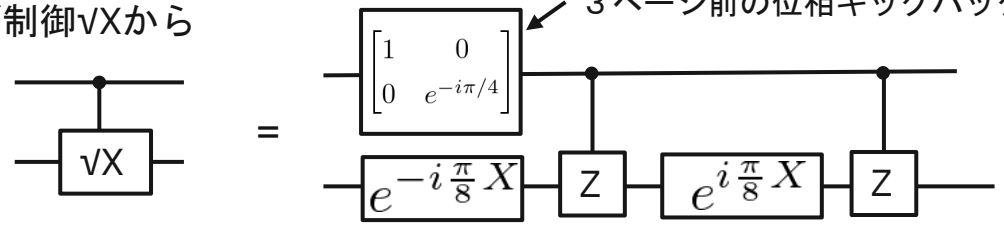
トフォリゲートは

U = X

V = √X

なので

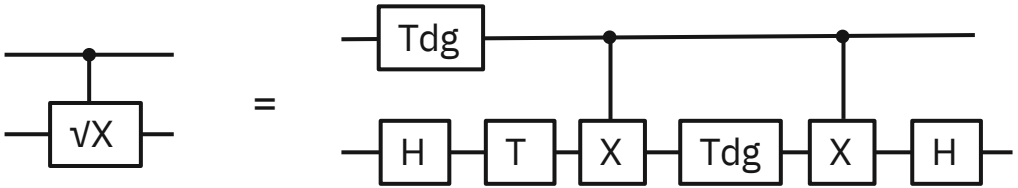
まず制御√Xから



$$X = e^{-i\frac{\pi}{2}} e^{-i\frac{\pi}{2}X}$$
$$\sqrt{X} = e^{-i\frac{\pi}{4}} e^{-i\frac{\pi}{4}X}$$
より

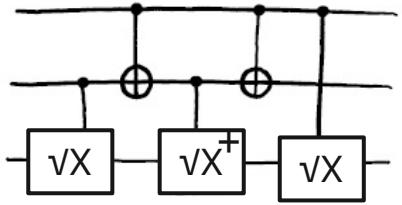
４ページ前の制御 $Rx(\theta)$より
制御Zと$Rx(\theta/2)$ で作れる制御 $Rx(\theta)$が作れる

$HXH = Z$ と同じく
$He^{-i\frac{\pi}{8}X}H = e^{-i\frac{\pi}{8}Z} = T$ を使って整理すると



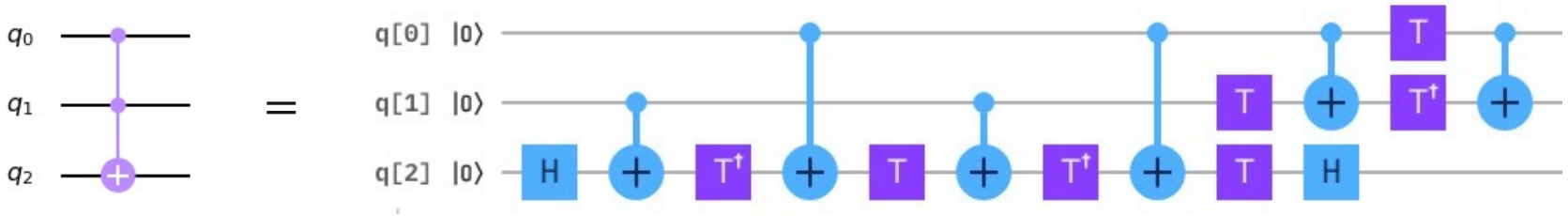
のようにT、CNOT、Hの回路になる。

この回路を



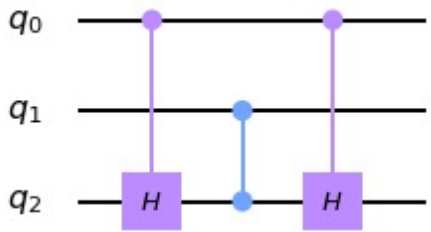
にいれるとT、CNOT、Hでトフォリ回路が作れる。
これを整理すると次ページ

T、CNOT、Hでのトフォリ回路



IQXで確認

相対位相が異なる他の回路例：
より少ないCNOTでゲートを実装できます。



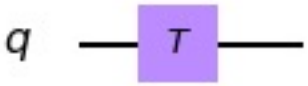
q0q1q2=|110>の場合のみ位相がπ

2.5-5 HとTによる任意の回転

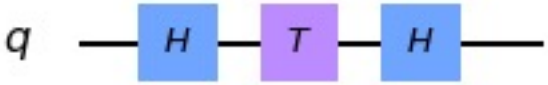
現在の量子ビットはノイズの影響を受けやすく、基本的には間違いを起こすゲートで構成されています。
（温度、磁場のゆらぎ、隣の量子ビットの動きなどが原因。）
大規模な量子アプリケーションでは、このノイズの影響を受けないように量子ビットをエンコードする必要があります。

単一量子ビットの回転$Rx(\theta)$、$Ry(\theta)$、$Rz(\theta)$は、角度θを完璧な精度で実装するのは不可能です。
これらの回転をフォールトトレラントな量子コンピューターに直接実装することはできません。

この章では、エラー訂正ができ、フォールトトレラントな量子コンピューターで、
HゲートとTゲートの効果が完全である場合に、
この二つのゲートを使って任意の回転θをよい近似で作れることを紹介します。

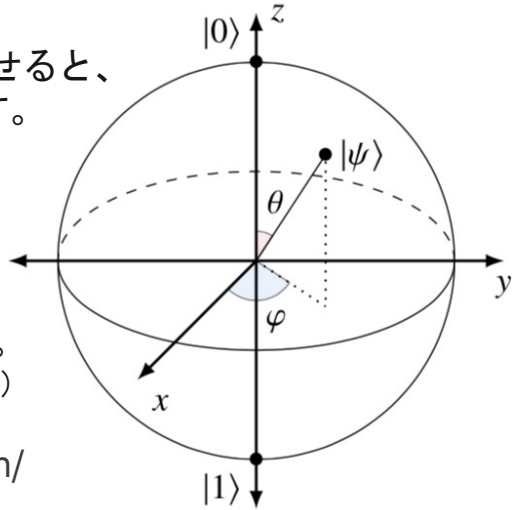


これはz軸を中心とした $\theta=\pi/4$ の回転なので、
数学的には $Rz(\pi/4)=e\exp(i\pi/8\ Z)$ と表されます。



Tゲートをアダマールで挟むとZ軸からX軸に変換されて
$Rx(\pi/4)$ ゲートができます。（X軸を中心とした$\pi/4$ の回転）

上記のあとにTゲートを組み合わせると、
$Rz(\pi/4)\ Rx(\pi/4)$ ゲートができます。



この$Rz(\pi/4)\ Rx(\pi/4)$ 回転は、その回転角が無理数になります。(証明なし)
n回繰り返すと、ある軸を中心に異なる角度で回転します。
無理数性より、異なる繰り返しから生じる角度は同じになることはありません。
（Tのみだと8回回転したら元に戻るが、TとHを組み合わせると無理数角度の回転になる）

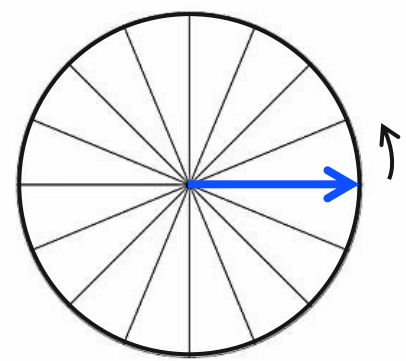
ブロッホ球で確認：https://javafxpert.github.io/grok-bloch/

このTHTH回転をたくさん繰り返すと任意の回転がよく近似できることを次ページで証明します。

（2.3章でも、X軸回転とZ軸回転があれば、任意の回転が近似できることを示しました。）

$Rz(\pi/4)\ Rx(\pi/4)$回転（THTH回転）をたくさん繰り返すと任意の回転がよりよく近似できることの証明

2πをn個で分割します。$Rz(\pi/4)\ Rx(\pi/4)$ 回転の各角度は0 から 2πの間のどこかです。



2πをn個にスライスのイメージ

回転角は、回転ごとに異なり、二度と同じ数にらないため（無理数なので）、回転のたびに、その角度はこれらのスライスのどこか1つに入ります。

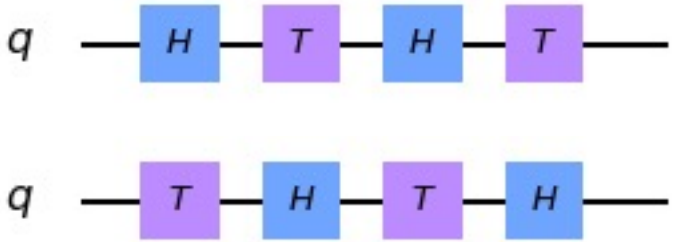
n回、回転すると、回転角は、それぞれどこかのスライスにばらばらに入ります（角度が偏る場合もありえます）。
スライスはn個あるので、n+1 回、回転すると、少なくとも1つのスライスに2つの角度が含まれます。

n以上のn1, n2について、
$n2-n1$回の回転の角度は、どこかのスライスにあり、かつ少なくとも2つの角度が含まれるので

$$\theta_{n_2-n_1} \neq 0, \quad -\frac{2\pi}{n} \leq \theta_{n_2-n_1} \leq \frac{2\pi}{n}.$$

したがって、nを増やすことで、小さな角度で回転させることができます。
このようにして、このゲートの繰り返す回数を増やすだけで、好きなだけ小さい角度で回転できます。

これは、最大$2\pi/n$まで正確であることが保証されます。



$Rz(\pi/4)$ $Rx(\pi/4)$ ゲート

$Rz(\pi/4)$と $Rx(\pi/4)$ の回転を逆の順序で行うと別の軸でも回転できます。

この二つの回路で2つの軸を中心とした任意の回転が得られ、
T ゲートのコストがかなりかかりますが、１量子ビットの任意の回転が実行できるようになります。

T ゲートが量子計算で非常に卓越しているのは、このように任意回転を作れるため。

実際、フォールトトレラントな量子コンピューターのアルゴリズムの複雑さは、T ゲートがいくつ必要かという点で議論されることがあります。
Tゲートを作るのは難しいので、できるだけ少ない T ゲートで回路を作ることが望まれます。

ここでの説明は、T ゲートをこのように任意回転作成に使用できることを示しただけで、
任意回転を作るために$Rz(\pi/4)$ $Rx(\pi/4)$ ゲートが最も効率的な方法とはいっていないことに注意してください。

2.3 位相キックバック

1. CNOTゲートの探索
2. 位相キックバック
   2.1 CNOT回路の等価性の説明
   2.2 Tゲートでのキックバック

2.4 普遍性の証明

1. はじめに
2. 普遍性の定義
3. 普遍性の証明
4. 普遍的量子ゲートのセット

2.5 基本的な回路の等価性

1. 制御ZをCNOTから作成する
2. 量子ビットのスワップ
3. 制御回転
4. トフォリゲート
5. HとTによる任意の回転

Thank you

Kifumi Numata

kifumi@jp.ibm.com

日本語訳『Qiskit Textbook』勉強会
第2章 2.3, 2.4, 2.5 振り返り

Kifumi Numata

振り返り

2.3 位相キックバック

1. CNOTゲートの探索
2. 位相キックバック
   2.1 CNOT回路の等価性の説明
   2.2 Tゲートでのキックバック

2.4 普遍性の証明

1. はじめに
2. 普遍性の定義
3. 普遍性の証明
4. 普遍的量子ゲートのセット

2.5 基本的な回路の等価性

1. 制御ZをCNOTから作成する
2. 量子ビットのスワップ
3. 制御回転
4. トフォリゲート
5. HとTによる任意の回転

2.3〜2.5章 振り返り

- 「2 .4章 普遍性の証明」では、3量子ビット以上の任意の回転を（スライスを使った証明で）Rx、H、CNOTゲートを使って作ることを説明。

$$U = e^{i(aX\otimes X\otimes X + bZ\otimes Z\otimes Z)}.$$

- 「2.5-5章 HとTによる任意の回転」では、1量子ビットをHTHT回転で作る（無理数角度の回転で）ことを説明。

- では、実際の量子コンピューターデバイスでは何を使っているのか？

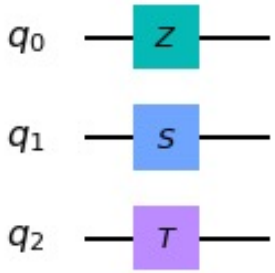
```
from qiskit import IBMQ
IBMQ.load_account()
ibmqx2 = IBMQ.get_provider('ibm-q').get_backend('ibmqx2')
ibmqx2.configuration().basis_gates
```

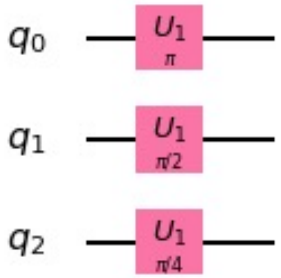
try

```
['u1', 'u2', 'u3', 'cx', 'id']
```

← ['u1', 'u2', 'u3', 'cx', 'id'] が基本ゲート

Transpilerを使ってみると `['u1', 'u2', 'u3', 'cx', 'id']` の回路になります。





$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = u1(\pi)$$

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} = u1(\pi/2)$$

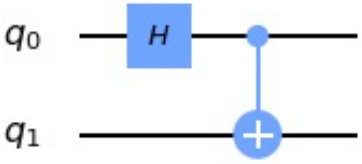
$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} = u1(\pi/4)$$

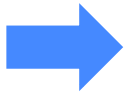
(Note)

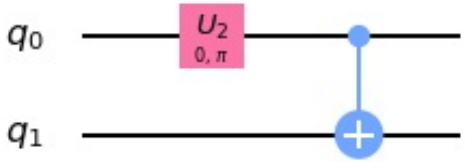
$$u3(\theta,\phi,\lambda) = U(\theta,\phi,\lambda) = \begin{pmatrix} \cos(\theta/2) & -e^{j\lambda}\sin(\theta/2) \\ e^{j\phi}\sin(\theta/2) & e^{j\lambda+j\phi}\cos(\theta/2) \end{pmatrix}$$

$$u1(\lambda) = U(0,0,\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{j\lambda} \end{pmatrix}$$

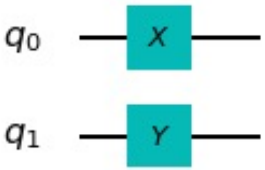
$$u2(\phi,\lambda) = U(\pi/2,\phi,\lambda) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -e^{j\lambda} \\ e^{j\phi} & e^{j\lambda+j\phi} \end{pmatrix}$$

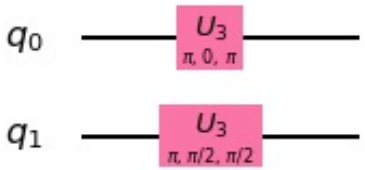






$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = u2(0,\pi)$$

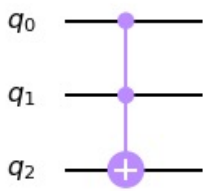




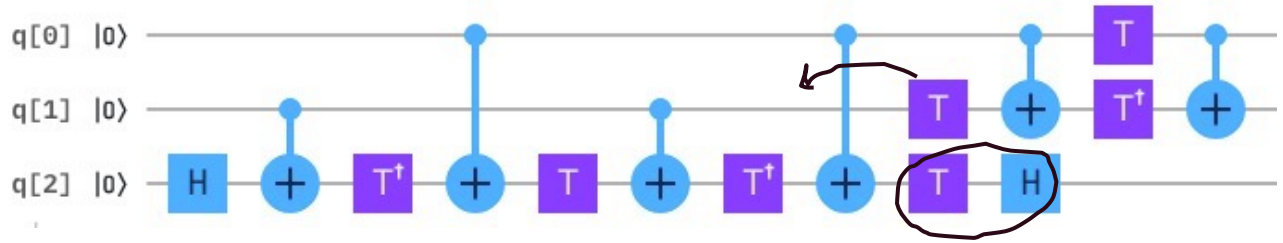
$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = u3(\pi,0,\pi)$$

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = u3(\pi,\pi/2,\pi/2)$$

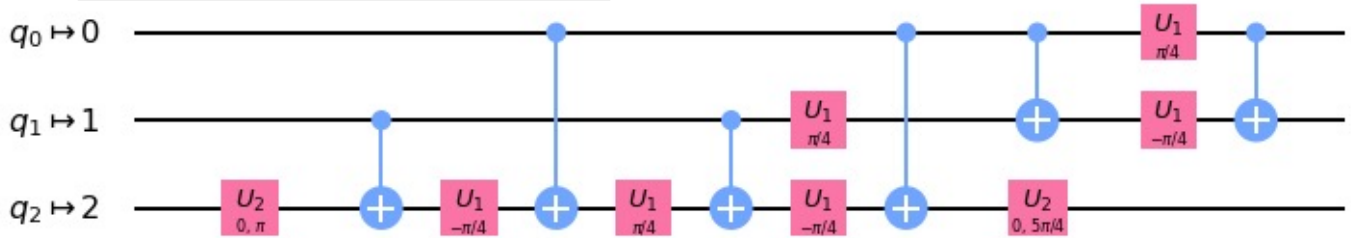
T、CNOT、Hでのトフォリ回路（2.5章で説明）



=



これも `['u1', 'u2', 'u3', 'cx', 'id']` にすると以下。

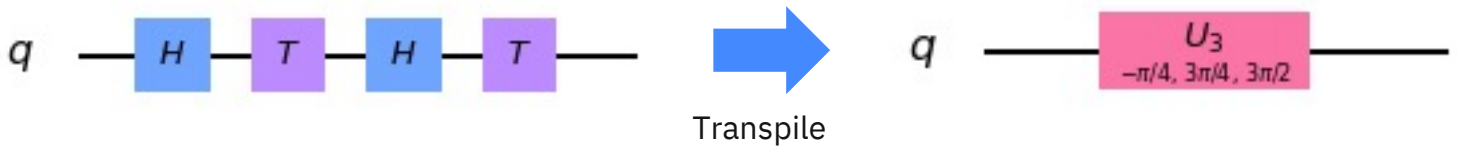


(Note)

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = u2(0,\pi)$$

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} = u1(\pi/4)$$

2.5章の任意の回転をつくる回路



(Note)

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = u2(0, \pi)$$

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} = u1(\pi/4)$$

$$u3(\theta, \phi, \lambda) = U(\theta, \phi, \lambda)$$

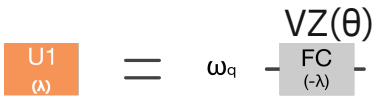
$$= \begin{pmatrix} \cos(\theta/2) & -e^{j\lambda}\sin(\theta/2) \\ e^{j\phi}\sin(\theta/2) & e^{j\lambda+j\phi}\cos(\theta/2) \end{pmatrix}$$

$$u1(\lambda) = U(0, 0, \lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{j\lambda} \end{pmatrix}$$

$$u2(\phi, \lambda) = U(\pi/2, \phi, \lambda)$$

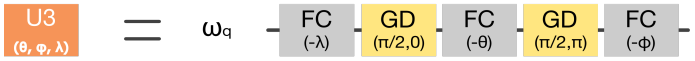
$$= \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -e^{j\lambda} \\ e^{j\phi} & e^{j\lambda+j\phi} \end{pmatrix}$$

実デバイスで実際に使用されているのは、Virtual-Z(θ)(Frame Change)と X(π/2) パルスとY(π/2)パルス

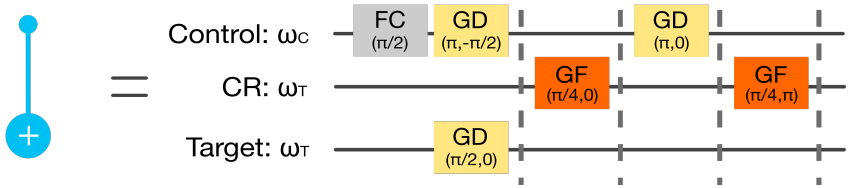


←U1(位相回転)はVirtual-Z(θ)(Frame Change) 。これはソフトウェアで行われる。
（ゲート実行時間ゼロ。）角度θは、ここで設定。

←U2はY(π/2)パルスの前後にFrame Change。
（1量子ビットのエラーはU2のエラーで測定されている）



←U3はX(π/2)パルスとX(-π/2)が必要（エラーはU2の2倍）。
U回転をZXZ分解して実装。



https://github.com/Qiskit/ibmq-device-information/tree/master/backends/yorktown/V1

Theorem 4.1: (*Z-Y* decomposition for a single qubit) Suppose U is a unitary operation on a single qubit. Then there exist real numbers α, β, γ and δ such that

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta). \quad (4.11)$$

参考：ニールセン&チャンの教科書のZYZ分解

←CNOTはパルス3つと共振パルス2つ。

⇨ ゲートが減ることはエラーが減ることにつながる

(Note) $u1(\lambda) = U(0,0,\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{j\lambda} \end{pmatrix}$

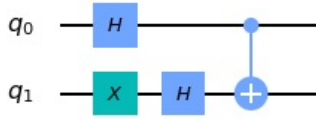
$$u2(\phi,\lambda) = U(\pi/2,\phi,\lambda) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{j\lambda} \\ e^{j\phi} & e^{j\lambda+j\phi} \end{pmatrix}$$

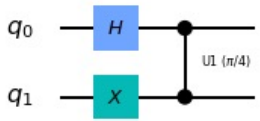
$$u3(\theta,\phi,\lambda) = U(\theta,\phi,\lambda) = \begin{pmatrix} \cos(\theta/2) & -e^{j\lambda}\sin(\theta/2) \\ e^{j\phi}\sin(\theta/2) & e^{j\lambda+j\phi}\cos(\theta/2) \end{pmatrix}$$

そのほか、2.3〜2.5章 振り返り

ゲートの等価性で学んだ変換は、今後のアルゴリズム勉強・作成に役立つだけでなく、
回路の最適化にも役立つ。

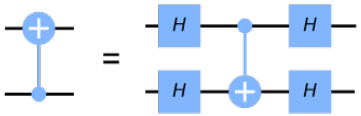
位相キックバック

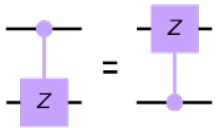


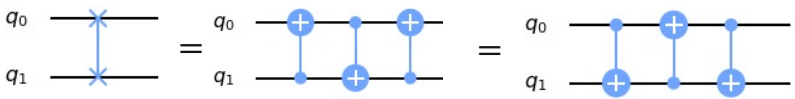


制御Tゲート

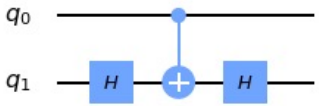
$$HZH = X.$$
$$HXH = Z$$







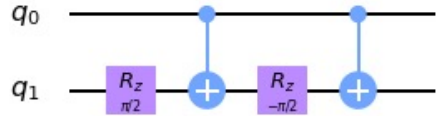
CZ =



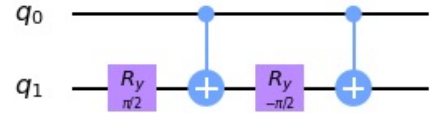
CY =

CH =

制御$Rz(\theta)$



制御$Ry(\theta)$



制御 $Rx(\theta)$