情報系の物理学 演習 7

G99P043-4
河邊昌彦

出題日　　:　2000/12/06
提出期限　:　2001/01/10
提出日　　:　2001/01/10

目 次

1 問題

演習 7 整数を二つの整数の積に分解する高速アルゴリズムを実現する、量子コンピュータの計算方法を説明せよ。

演習 7' 量子コンピュータによる計算を適当な具体例を一つあげて、それを元に説明せよ。

2 概略

量子コンピュータを動作させるということは、即ち、内部状態にユニタリ作用素を作用させるということである。また、その結果を観測するということは、エルミート作用素の固有値の一つをある確率分布に従って得ることであり、その観測によって、状態が対応する固有空間へと射影される。原理的にはこれで正しいのであるが、このままでは具体的過ぎるため、アルゴリズムを考える上では不向きである。そこで、まず、テンソル積の性質などを使って量子コンピュータの動作を抽象化する。そして、その上で各種アルゴリズムを解説し、さらにシミュレータによって具体的な動作を見ることにする。

なお、因数分解のアルゴリズムに関しては [1] を、量子コンピュータのゲートに関しては [2] を、シミュレータの実現に関しては [3] を参考にした。

3 量子コンピュータの抽象化

3.1 ベクトルとテンソル積

量子コンピュータの状態は、テンソル積ベクトル空間の要素で表されている。ここで、

$$\begin{aligned} V &: m \text{ 次元テンソル積ベクトル空間} \\ W &: n \text{ 次元テンソル積ベクトル空間} \end{aligned}$$

とし、

$$\begin{aligned} |x_1\rangle, |x_2\rangle &\in V \\ |y_1\rangle, |y_2\rangle &\in W \end{aligned}$$

とする。具体的には

$$|x_1\rangle = \sum_{i=0}^{m-1} x_{1i} |i\rangle \qquad (|i\rangle \text{ は } i \text{ 番目の標準基底})$$

等となっている。これを用いると、ベクトルのテンソル積は

$$\begin{aligned} |x_1\rangle \otimes |y_1\rangle &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{1i} \cdot y_{1j} |i\rangle \otimes |j\rangle \\ &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{1i} \cdot y_{1j} |i \cdot n + j\rangle \end{aligned}$$

となる。また、ベクトルの和とスカラー倍に関してテンソル積には次のような性質がある。

定理 1

$$
\begin{aligned}
(|x_1\rangle + |x_2\rangle) \otimes |y_1\rangle &= |x_1\rangle \otimes |y_1\rangle + |x_2\rangle \otimes |y_1\rangle && (1)\\
|x_1\rangle \otimes (|y_1\rangle + |y_2\rangle) &= |x_1\rangle \otimes |y_1\rangle + |x_1\rangle \otimes |y_2\rangle && (2)\\
(\alpha\,|x_1\rangle) \otimes |y_1\rangle = |x_1\rangle \otimes (\alpha\,|y_1\rangle) &= \alpha(|x_1\rangle \otimes |y_1\rangle) \qquad (\alpha \in \mathbf{C}) && (3)
\end{aligned}
$$

証明

(1)

$$
\begin{aligned}
(|x_1\rangle + |x_2\rangle) \otimes |y_1\rangle &= \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}(x_{1i} + x_{2i})y_{1j}\,|i\rangle \otimes |j\rangle \\
&= \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}(x_{1i}y_{1j} + x_{2i}y_{1j})\,|i\rangle \otimes |j\rangle \\
&= \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}x_{1i}y_{1j}\,|i\rangle \otimes |j\rangle + \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}x_{2i}y_{1j}\,|i\rangle \otimes |j\rangle \\
&= |x_1\rangle \otimes |y_1\rangle + |x_2\rangle \otimes |y_1\rangle
\end{aligned}
$$

(2)

$$
\begin{aligned}
|x_1\rangle \otimes (|y_1\rangle + |y_2\rangle) &= \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}x_{1i}(y_{1j} + y_{2j})\,|i\rangle \otimes |j\rangle \\
&= \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}x_{1i}y_{1j}\,|i\rangle \otimes |j\rangle + \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}x_{1i}y_{2j}\,|i\rangle \otimes |j\rangle \\
&= |x_1\rangle \otimes |y_1\rangle + |x_1\rangle \otimes |y_2\rangle
\end{aligned}
$$

(3)

$$
\begin{aligned}
(\alpha\,|x_1\rangle) \otimes |y_1\rangle &= \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}(\alpha x_{1i})y_{1j}\,|i\rangle \otimes |j\rangle \\
&= \alpha\sum_{i=0}^{n-1}\sum_{j=0}^{m-1}x_{1i}y_{1j}\,|i\rangle \otimes |j\rangle \\
&= \alpha(|x_1\rangle \otimes |y_1\rangle)
\end{aligned}
$$

$$
\begin{aligned}
|x_1\rangle \otimes (\alpha\,|y_1\rangle) &= \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}x_{1i}(\alpha y_{1j})\,|i\rangle \otimes |j\rangle \\
&= \alpha\sum_{i=0}^{n-1}\sum_{j=0}^{m-1}x_{1i}y_{1j}\,|i\rangle \otimes |j\rangle \\
&= \alpha(|x_1\rangle \otimes |y_1\rangle)
\end{aligned}
$$

3.2 行列とテンソル積

行列 A_1, A_2, B_1, B_2 を

$$\begin{aligned} A_1, A_2 &: V \longrightarrow V \\ B_1, B_2 &: W \longrightarrow W \end{aligned}$$

とする。即ち、A_1, A_2 は m 次正方行列、B_1, B_2 は n 次正方行列である。これらの行列を次のように要素によって表すことにする。

$$A_1 = \sum_{i_1=0}^{m-1} \sum_{i_2=0}^{m-1} a_{1i_1i_2} |i_1\rangle \langle i_2| = \sum_{i_1,i_2=0}^{m-1} a_{1i_1i_2} |i_1\rangle \langle i_2|$$

このようにすると、行列の積とテンソル積は以下のようになる。

$$\begin{aligned} A_1A_2 &= \sum_{i_1,i_2=0}^{m-1} \left(\sum_{j=0}^{m-1} a_{1i_1j} \cdot a_{2ji_2} \right) |i_1\rangle \langle i_2| \\ A_1 \otimes B_1 &= \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2} \cdot b_{1j_1j_2} (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \end{aligned}$$

行列に関しても、ベクトルと同様にテンソル積に関する性質がある。

定理 2

$$(A_1 + A_2) \otimes B_1 = A_1 \otimes B_1 + A_2 \otimes B_1 \tag{4}$$

$$A_1 \otimes (B_1 + B_2) = A_1 \otimes B_1 + A_1 \otimes B_2 \tag{5}$$

$$(\alpha A_1) \otimes B_1 = A_1 \otimes (\alpha B_1) = \alpha (A_1 \otimes B_1) \tag{6}$$

$$(A_1A_2) \otimes (B_1B_2) = (A_1 \otimes B_1)(A_2 \otimes B_2) \tag{7}$$

$$(A_1 |x_1\rangle) \otimes (B_1 |y_1\rangle) = (A_1 \otimes B_1)(|x_1\rangle \otimes |y_1\rangle) \tag{8}$$

$$A_1{}^* \otimes B_1{}^* = (A_1 \otimes B_1)^* \tag{9}$$

$$A_1A_1{}^* = I_m \wedge B_1B_1{}^* = I_n \Rightarrow (A_1 \otimes B_1)(A_1 \otimes B_1)^* = I_{mn} \tag{10}$$

$$A_1 = A_1{}^* \wedge B_1 = B_1{}^* \Rightarrow A_1 \otimes B_1 = (A_1 \otimes B_1)^* \tag{11}$$

ただし、I_m は m 次単位行列とする。

証明

(4)

$$\begin{aligned} (A_1 + A_2) \otimes B_1 &= \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} (a_{1i_1i_2} + a_{2i_1i_2}) b_{1j_1j_2} (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\ &= \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} (a_{1i_1i_2} b_{1j_1j_2} + a_{2i_1i_2} b_{1j_1j_2}) (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\ &= \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2} b_{1j_1j_2} (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\ &\quad + \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{2i_1i_2} b_{1j_1j_2} (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\ &= A_1 \otimes B_1 + A_2 \otimes B_1 \end{aligned}$$

(5)

$$\begin{aligned}
A_1 \otimes (B_1 + B_2) &= \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2}(b_{1j_1j_2} + b_{2j_1j_2})(|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\
&= \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2} b_{1j_1j_2}(|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\
&\quad + \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2} b_{2j_1j_2}(|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\
&= A_1 \otimes B_1 + A_1 \otimes B_2
\end{aligned}$$

(6)

$$\begin{aligned}
(\alpha A_1) \otimes B_1 &= \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} (\alpha a_{1i_1i_2}) b_{1j_1j_2}(|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\
&= \alpha \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2} b_{1j_1j_2}(|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\
&= \alpha(A_1 \otimes B_1)
\end{aligned}$$

$$\begin{aligned}
A_1 \otimes (\alpha B_1) &= \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2} (\alpha b_{1j_1j_2})(|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\
&= \alpha \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2} b_{1j_1j_2}(|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\
&= \alpha(A_1 \otimes B_1)
\end{aligned}$$

(7)

$$\begin{aligned}
(A_1A_2) \otimes (B_1B_2) &= \left(\sum_{i_1,i_2=0}^{m-1} \sum_{i_3=0}^{m-1} a_{1i_1i_3} a_{2i_3i_2} |i_1\rangle \langle i_2| \right) \otimes \left(\sum_{j_1,j_2=0}^{n-1} \sum_{j_3=0}^{n-1} b_{1j_1j_3} b_{2j_3j_2} |j_1\rangle \langle j_2| \right) \\
&= \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} \left(\sum_{i_3=0}^{m-1} a_{1i_1i_3} a_{2i_3i_2} \right) \left(\sum_{j_3=0}^{n-1} b_{1j_1j_3} b_{2j_3j_2} \right) (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\
&= \sum_{i_1,i_2,i_3=0}^{m-1} \sum_{j_1,j_2,j_3=0}^{n-1} a_{1i_1i_3} a_{2i_3i_2} b_{1j_1j_3} b_{2j_3j_2}(|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|)
\end{aligned}$$

$$
\begin{aligned}
&(A_1 \otimes B_1)(A_2 \otimes B_2) \\
&= \left(\sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2} b_{1j_1j_2} (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \right) \\
&\quad \cdot \left(\sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{2i_1i_2} b_{2j_1j_2} (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \right) \\
&= \left(\sum_{i_1=0}^{m-1} \sum_{j_1=0}^{n-1} \sum_{i_2=0}^{m-1} \sum_{j_2=0}^{n-1} a_{1i_1i_2} b_{1j_1j_2} (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \right) \\
&\quad \cdot \left(\sum_{i_1=0}^{m-1} \sum_{j_1=0}^{n-1} \sum_{i_2=0}^{m-1} \sum_{j_2=0}^{n-1} a_{2i_1i_2} b_{2j_1j_2} (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \right) \\
&= \sum_{i_1=0}^{m-1} \sum_{j_1=0}^{n-1} \sum_{i_2=0}^{m-1} \sum_{j_2=0}^{n-1} \left(\sum_{i_3=0}^{m-1} \sum_{j_3=0}^{n-1} a_{1i_1i_3} b_{1j_1j_3} \cdot a_{2i_3i_2} b_{2j_3j_2} \right) (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\
&= \sum_{i_1,i_2,i_3=0}^{m-1} \sum_{j_1,j_2,j_3=0}^{n-1} a_{1i_1i_3} a_{2i_3i_2} b_{1j_1j_3} b_{2j_3j_2} (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|)
\end{aligned}
$$

$$
\therefore (A_1 A_2) \otimes (B_1 B_2) \quad = \quad (A_1 \otimes B_1)(A_2 \otimes B_2)
$$

(8)

$$
\begin{aligned}
(A_1 |x_1\rangle) \otimes (B_1 |y_1\rangle) \quad &= \quad \left(\sum_{i_1=0}^{m-1} \left(\sum_{i_2=0}^{m-1} a_{1i_1i_2} x_{1i_2} \right) |i_1\rangle \right) \otimes \left(\sum_{j_1=0}^{n-1} \left(\sum_{j_2=0}^{n-1} b_{1j_1j_2} y_{1j_2} \right) |j_1\rangle \right) \\
&= \quad \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2} x_{1i_2} b_{1j_1j_2} y_{1j_2} \, |i_1\rangle \otimes |i_2\rangle
\end{aligned}
$$

$$
\begin{aligned}
&(A_1 \otimes B_1)(|x_1\rangle \otimes |y_1\rangle) \\
&= \left(\sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2} b_{1i_1i_2} (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \right) \cdot \left(\sum_{i_2=0}^{m-1} \sum_{j_2=0}^{n-1} x_{1i_2} y_{1j_2} \, |i_2\rangle \otimes |j_2\rangle \right) \\
&= \left(\sum_{i_1=0}^{m-1} \sum_{j_1=0}^{n-1} \sum_{i_2=0}^{m-1} \sum_{j_2=0}^{n-1} a_{1i_1i_2} b_{1i_1i_2} (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \right) \cdot \left(\sum_{i_2=0}^{m-1} \sum_{j_2=0}^{n-1} x_{1i_2} y_{1j_2} \, |i_2\rangle \otimes |j_2\rangle \right) \\
&= \sum_{i_1=0}^{m-1} \sum_{j_1=0}^{n-1} \left(\sum_{i_2=0}^{m-1} \sum_{j_2=0}^{n-1} a_{1i_1i_2} b_{1i_1i_2} \cdot x_{1i_2} y_{1j_2} \right) |i_1\rangle \otimes |j_1\rangle \\
&= \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2} x_{1i_2} b_{1j_1j_2} y_{1j_2} \, |i_1\rangle \otimes |i_2\rangle
\end{aligned}
$$

$$
\therefore (A_1 |x_1\rangle) \otimes (B_1 |y_1\rangle) \quad = \quad (A_1 \otimes B_1)(|x_1\rangle \otimes |y_1\rangle)
$$

(9)

$$\begin{aligned}
{A_1}^* \otimes {B_1}^* &= \left(\sum_{i_1,i_2=0}^{m-1} {a_{1i_2i_1}}^* |i_1\rangle \langle i_2| \right) \otimes \left(\sum_{j_1,j_2=0}^{n-1} {b_{1j_2j_1}}^* |j_1\rangle \langle j_2| \right) \\
&= \sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} {a_{1i_2i_1}}^* \cdot {b_{1j_2j_1}}^* (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \\
&= \left(\sum_{i_1,i_2=0}^{m-1} \sum_{j_1,j_2=0}^{n-1} a_{1i_1i_2} b_{1j_1j_2} (|i_1\rangle \otimes |j_1\rangle)(\langle i_2| \otimes \langle j_2|) \right)^* \\
&= (A_1 \otimes B_1)^*
\end{aligned}$$

(10)

$$\begin{aligned}
(A_1 \otimes B_1)(A_1 \otimes B_1)^* &= (A_1 \otimes B_1)({A_1}^* \otimes {B_1}^*) \\
&= (A_1 {A_1}^*) \otimes (B_1 {B_1}^*) \\
&= I_m \otimes I_n \\
&= I_{mn}
\end{aligned}$$

(11)

$$\begin{aligned}
A_1 \otimes B_1 &= {A_1}^* \otimes {B_1}^* \\
&= (A_1 \otimes B_1)^*
\end{aligned}$$

3.3 状態の推移

量子コンピュータの状態は、ユニタリ作用素によって推移する。即ち、ユニタリ作用素 U によって

$$|x\rangle \longmapsto U|x\rangle$$

と状態 $|x\rangle$ が変化する。ここで、基底の線形結合で $|x\rangle$ を表すと

$$\begin{aligned}
U|x\rangle &= U \sum_{i=0}^{n-1} x_i |i\rangle \\
&= \sum_{i=0}^{n-1} x_i U |i\rangle
\end{aligned}$$

となる。基底は常にテンソル積に分解できるので、$|i\rangle = |i_1\rangle \otimes |i_2\rangle$ とできる。そして、$U = U_1 \otimes U_2$, と表されるならば、

$$\begin{aligned}
\sum_{i=0}^{n-1} x_i U |i\rangle &= \sum_{i=0}^{n-1} x_i (U_1 \otimes U_2)(|i_1\rangle \otimes |i_2\rangle) \\
&= \sum_{i=0}^{n-1} x_i (U_1 |i_1\rangle) \otimes (U_2 |i_2\rangle)
\end{aligned}$$

となる。これは即ち、基底をテンソル積に分解し、その各々に独立してユニタリ作用素を作用させることができる、ということを示している。特に、$U_2 = I$ と考えれば、基底に対して、部分的にユニタリ作用素を作用させられる、ということになる。

さらに、次のようなユニタリ作用素を考える。

$$\begin{aligned} S_1 &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ S_{ij} &= I_i \otimes S_1 \otimes I_j \end{aligned}$$

S_1 は次のようにテンソル積のオペランドを交換する作用素である。

$$\begin{aligned} S_1(|0\rangle \otimes |0\rangle) &= {S_1}^t(1,0,0,0) = {}^t(1,0,0,0) = (|0\rangle \otimes |0\rangle) \\ S_1(|0\rangle \otimes |1\rangle) &= {S_1}^t(0,1,0,0) = {}^t(0,0,1,0) = (|1\rangle \otimes |0\rangle) \\ S_1(|1\rangle \otimes |0\rangle) &= {S_1}^t(0,0,1,0) = {}^t(0,1,0,0) = (|0\rangle \otimes |1\rangle) \\ S_1(|1\rangle \otimes |1\rangle) &= {S_1}^t(0,0,0,1) = {}^t(0,0,0,1) = (|1\rangle \otimes |1\rangle) \end{aligned}$$

まとめると、$|q_0\rangle, |q_1\rangle \in \mathbf{C^2}$ として、

$$S_1(|q_0\rangle \otimes |q_1\rangle) = |q_1\rangle \otimes |q_0\rangle$$

ということである。そして、S_{ij} は $(i+j+2)$ 次元の基底 $|b\rangle = |q_0\rangle \otimes \cdots \otimes |q_{i+j+1}\rangle$ に対して、

$$\begin{aligned} S_{ij}|b\rangle &= (I_i \otimes S_1 \otimes I_j)(|q_0\rangle \otimes \cdots \otimes |q_{i+j+1}\rangle) \\ &= (I_i(|q_0\rangle \otimes \cdots \otimes |q_{i-1}\rangle)) \otimes (S_1(|q_i\rangle \otimes |q_{i+1}\rangle)) \otimes (I_j(|q_{i+2}\rangle \otimes \cdots \otimes |q_{i+j-1}\rangle)) \\ &= |q_0\rangle \otimes \cdots \otimes |q_{i-1}\rangle \otimes |q_{i+1}\rangle \otimes |q_i\rangle \otimes |q_{i+2}\rangle \otimes \cdots \otimes |q_{i+j-1}\rangle \end{aligned}$$

と、$|q_i\rangle$ と $|q_{i+1}\rangle$ を交換する。従って、S_ij を何度も繰り返して作用させることで、$|q_0\rangle, \cdots, |q_{i+j+1}\rangle$ を任意の順番に並べ替えることができる。

以上の二つの議論をまとめると、基底 $|b\rangle = |q_0\rangle \otimes \cdots \otimes |q_{n-1}\rangle$ に対し、任意個の $|q_{i_1}\rangle, \cdots, |q_{i_m}\rangle$ を取り出し、任意に並べ替えてから、ユニタリ作用素を作用させることができる、ということである。また、量子コンピュータの状態空間は 2^n 次元であるので、基底 $|b\rangle$ を最も細かくテンソル積に分解すると、$|q_i\rangle \in \{|0\rangle, |1\rangle\}$ である。即ち、各 $|q_i\rangle$ をビット (量子ビット;qubit) と見なすことができる。そのように見ると、以上の議論は、任意の位置のビットに対してユニタリ作用素を作用させられる、ということである。

3.4 観測

量子コンピュータにおける観測は、エルミート作用素 A の固有値を確率的に得て、それに対応する固有空間へ状態が射影されることである。ここで、任意のエルミート作用素 A に対してはあるユニタリ作用素 L が存在して、

$$\begin{aligned} L^{-1}AL = L^*AL &= D \\ D &= \sum_{i=0}^{n-1} d_i |i\rangle \langle i| \end{aligned}$$

と対角化可能である ([4] p.163)。また、A の固有値を $\lambda_1, \cdots, \lambda_r$ とし、各固有空間への正射影をを $P_1, \cdots, P_r$ とすると、A は次のようにスペクトル分解できる ([4] p.164)。

$$\begin{aligned}
\sum_{i=1}^{r} P_i &= I_n \\
P_i P_j &= \delta_{ij} P_i \qquad (\forall i, j) \\
{P_i}^* &= P_i \\
A &= \sum_{i=1}^{r} \lambda_i P_i
\end{aligned}$$

よって、

$$\begin{aligned}
D &= L^* A L \\
&= L^* \left(\sum_{i=1}^{r} \lambda_i P_i \right) L \\
&= \sum_{i=1}^{r} \lambda_i L^* P_i L
\end{aligned}$$

となる。ここで、$L^* P_i L$ は、

$$\begin{aligned}
&\sum_{i=1}^{r} L^* P_i L = L^* \left(\sum_{i=1}^{r} P_i \right) L = L^* I_n L = L^* L = I_n \\
&(L^* P_i L)(L^* P_j L) = L^* P_i L L^* P_j L = L^* P_i P_j L = L^* (\delta_{ij} P_i) L = \delta_{ij} (L^* P_i L) \\
&(L^* P_i L)^* = L^* {P_i}^* (L^*)^* = L^* P_i L
\end{aligned}$$

であるので、D の正射影によるスペクトル分解となっている。よって、D の固有値も $\lambda_1, \cdots, \lambda_r$ である。この $L^* P_i L$ を ${P_i}'$ と置く。

$$\begin{aligned}
L^* P_i L &= {P_i}' \\
P_i &= L {P_i}' L^*
\end{aligned}$$

P_i によるベクトル $|x\rangle$ の正射影は、次のようになる。

$$\begin{aligned}
P_i |x\rangle &= L {P_i}' L^* |x\rangle \\
|P_i |x\rangle| &= |L {P_i}' L^* |x\rangle| = |{P_i}' L^* |x\rangle|
\end{aligned}$$

二番目の式の変形は、L がユニタリ行列であるため、最後の L の作用ではベクトルの大きさは変わらないことによる。以上から、$|x\rangle$ という状態においては、

$$\begin{aligned}
A \text{ を観測} &\iff \text{確率 } |P_i |x\rangle|^2 \text{ で } \lambda_i \text{ が得られ、状態が } \tfrac{P_i|x\rangle}{|P_i|x\rangle|} \text{ に移る} \\
&\iff \text{確率 } |{P_i}' L^* |x\rangle|^2 \text{ で } \lambda_i \text{ が得られ、状態が } L \tfrac{{P_i}' L^*|x\rangle}{|{P_i}' L^*|x\rangle|} \text{ に移る} \\
&\iff L^* \text{ を作用してから } D \text{ を観測し、その後 } L \text{ を作用}
\end{aligned}$$

ということが成り立つ。即ち、任意の A という観測量に対して、A の観測は、ユニタリ作用素の作用と対角行列で表される観測量の観測で、置き換え可能である。従って、観測量として、対角行列で表現される作用素だけを考えても、一般性は失われない。

そこで、ここからは観測量 A は、

$$A \quad = \quad \sum_{i=0}^{n-1} a_i \left|i\right\rangle \left\langle i\right|$$

であるとする。A の固有値は

$$\{\lambda_1, \cdots, \lambda_r\} \quad = \quad \{a_0, \cdots, a_{n-1}\}$$

であり、スペクトル分解 $P_1, \cdots, P_r$ は

$$P_i \quad = \quad \sum_{j:a_j=\lambda_i} \left|j\right\rangle \left\langle j\right|$$

ととれる。ここで、二つの観測量 A, A' のスペクトル分解が、共に $P_1, \cdots, P_r$ であったとする。A, A' の固有値をそれぞれ $\lambda_1, \cdots, \lambda_r,\ \lambda_1{}', \cdots, \lambda_r{}'$ とすると、

$$\begin{aligned} \lambda_i \text{ が得られる確率} = \quad & \left|P_i \left|x\right\rangle\right|^2 \quad = \lambda_i{}' \text{ が得られる確率} \\ \lambda_i \text{ が得られた場合の状態推移} = \quad & P_i \left|x\right\rangle \quad = \lambda_i{}' \text{ が得られた場合の状態推移} \end{aligned}$$

である。また、$\{\lambda_1, \cdots, \lambda_r\}$ と $\{\lambda_1{}', \cdots, \lambda_r{}'\}$ の間には $\varphi(\lambda_i) = \lambda_i{}'$ という全単射が存在する。よって、A と A' は観測量としては等価であると言うことができるので、観測量の性質はそのスペクトル分解 $P_1, \cdots, P_r$ で決定されることになる。そして、実際に i 番目の固有値が得られたならば、P_i を求めることができる。以上から、観測とは、P_i が確率 $\left|P_i \left|x\right\rangle\right|^2$ で得られ、状態が $\frac{P_i\left|x\right\rangle}{\left|P_i\left|x\right\rangle\right|}$ に移ることである、と言い換えられる。

さらに P_i に関しては、$P_1, \cdots, P_r$ が $\left|0\right\rangle, \cdots, \left|n-1\right\rangle$ を類別していると見ることができる。即ち、$Q_1, \cdots, Q_r$ に対して

$$\begin{aligned} Q_i \quad &\subset \quad U = \{\left|0\right\rangle, \cdots, \left|n-1\right\rangle\} \\ Q_i \quad &\neq \quad \emptyset \\ Q_i \cap Q_j \quad &= \quad \emptyset \qquad (i \neq j) \\ \bigcup_i Q_i \quad &= \quad U \end{aligned}$$

とすれば、P_i は

$$P_i \quad = \quad \sum_{\left|j\right\rangle \in Q_i} \left|j\right\rangle \left\langle j\right|$$

と決定できる。よって、

$$P_i \quad = \quad P(Q_i)$$

と書くことにし、以下では $U = \bigcup_i Q_i$ という類別の方法に関して論じる。

集合を類別するには、同値関係 “$\sim$” を定義し、それによって同値類に分ければ良い。例えば、

$$\left|i\right\rangle \sim \left|j\right\rangle \quad : \quad \left|i\right\rangle = \left|j\right\rangle$$

と定義すると、U は

$$U \quad = \quad \bigcup_i \{\left|i\right\rangle\}$$

と類別される。特にここでは次のような類別を取り上げる。即ち

$$(|i_1\rangle \otimes |i_2\rangle) \sim (|j_1\rangle \otimes |j_2\rangle) \quad : \quad |i_1\rangle = |j_1\rangle \qquad (|i_1\rangle, |j_1\rangle \in \mathbf{C}^{2^m}, |i_2\rangle, |j_2\rangle \in \mathbf{C}^{2^n})$$

として、

$$\begin{aligned} U &= \bigcup_{i=0}^{2^m-1} Q_i \\ Q_i &= \bigcup_{j=0}^{2^n-1} \{|i\rangle \otimes |j\rangle\} \end{aligned}$$

と、$Q_0, \cdots, Q_{2^m-1}$ に類別する。この類別は上位 m ビットによる類別である。ここで、

$$\begin{aligned} P(Q_i) &= \sum_{j=0}^{2^n-1} P(\{|i\rangle \otimes |j\rangle\}) \\ |P(Q_i)\,|x\rangle\,|^2 &= \sum_{j=0}^{2^n-1} |P(\{|i\rangle \otimes |j\rangle\})\,|x\rangle\,|^2 \end{aligned}$$

となっているので、射影 $P(Q_i)$ によって下位 n ビットは保存され、また、確率分布は上位 m ビットの周辺確率分布となる。これは即ち、上位 m ビットだけを観測した、ということである。

以上から、ビット数を限定して観測することができ、また、前節の議論からビット順を任意に並べ替えることができる。従ってまとめると、任意個のビットを任意の位置から選んで、そこだけを観測できる、ということになる。

3.5 ゲートアレイ

ここまでの議論で、ユニタリ作用素の作用及び観測は、自由にビットを選んで行えることが分かった。そこで、各量子ビットを “配線” で表し、作用素を “ゲート” で表すことで、古典コンピュータにおけるゲートアレイに相当するものを書くことができる。ただし、作用素は正方行列で表されるので、ゲートの入力ビット数と出力ビット数は常に等しくなる。また、ファンアウトやファンイン、フィードバックなどはなく、回路は常に直線的である。

例えば、図 1 は、2 ビット目と 3 ビット目に対して 4 次元のユニタリ作用素 U を作用させる回路である。これを式で書けば

$$|q_3', q_2', q_1', q_0'\rangle \quad = \quad |q_3\rangle \otimes U\,|q_2, q_1\rangle \otimes |q_0\rangle$$

ということである。

以下では基本的なゲートに関して説明する。

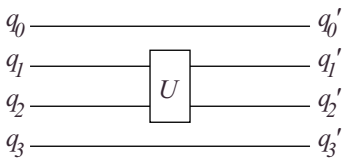

図 1: 4 ビット回路

3.5.1 NOT ゲート

NOT ゲートは 1 ビットのゲートであり、

$$\begin{aligned} |0\rangle &\longmapsto |1\rangle \\ |1\rangle &\longmapsto |0\rangle \end{aligned}$$

と変化させる。行列で表せば

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

である。以後これを σ_x と表す。例えば、全てのビットを反転させる回路は図 2 のようになる。

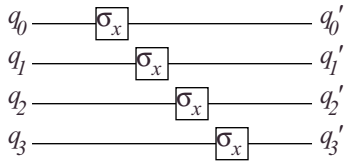

図 2: NOT ゲート

3.5.2 $\sqrt{\text{NOT}}$ゲート

$\sqrt{\text{NOT}}$ゲートは 1 ビットゲートであり、純粋状態から重ね合わせの状態を作り出す。即ち、

$$\begin{aligned} |0\rangle &\longmapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |1\rangle &\longmapsto \frac{1}{\sqrt{2}}(-|0\rangle + |1\rangle) \end{aligned}$$

という変換であり、行列で表せば、

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

である。これを用いて、図 3 のような回路を作り、入力として $|0000\rangle$ という状態を入れると、

$$\frac{1}{4}\sum_{i_0,i_1,i_2,i_3=0}^{1} |i_3, i_2, i_1, i_0\rangle$$

という状態に推移する。

3.5.3 二次元ユニタリゲート

古典コンピュータのゲートにおいては、NAND によって全ての論理回路を構成することができた。それと同様に、[2] に依れば、全てのユニタリ作用素は 2 種類のゲートのみで構成可能である。そのうちの一つが二次元ユニタリゲートであり、実際には二次元ユニタリゲートの中から一つだけを選べば良い。二次元ユニタリゲートは 1 ビットゲートであり、NOT ゲート、$\sqrt{\text{NOT}}$ゲートもこれに含まれる。

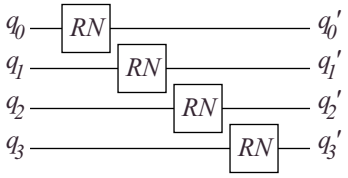

図 3: $\sqrt{\text{NOT}}$ゲート (RN は$\sqrt{\text{NOT}}$を表す)

3.5.4 制御 NOT ゲート

二次元ユニタリゲートと共にユニバーサルなゲートを構成するのが制御 NOT ゲートである。制御 NOT ゲートは 2 ビットゲートであり、1 ビット目で 2 ビット目の NOT を制御する。即ち、

$$|q_1, q_0\rangle \quad \longmapsto \quad \begin{cases} |q_1, \neg q_0\rangle & (q_1 = 1) \\ |q_1, q_0\rangle & (q_1 = 0) \end{cases}$$

というゲートである。正確に書くと、

$$\begin{aligned} |00\rangle &\longmapsto |00\rangle \\ |01\rangle &\longmapsto |01\rangle \\ |10\rangle &\longmapsto |11\rangle \\ |11\rangle &\longmapsto |10\rangle \end{aligned}$$

であるので、この表現行列は

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

となる。回路図上では、反転されるビットに σ_x を書き、制御ビットに制御線を引く。ユニバーサルなゲートであるため、このゲートを使った回路の例は多くあるが、例えば図 4 は以前にも出てきたビットスワップゲートである。

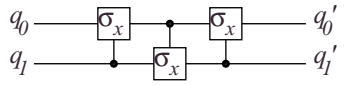

図 4: ビットスワップ回路

或いは、制御 NOT ゲートは XOR ゲートと見ることもできる。即ち、

$$|q_1, q_0\rangle \quad \longmapsto \quad |q_1, q_1 \oplus q_0\rangle$$

ということである。

3.5.5 制御ユニタリゲート

以上までの基本ゲートで全ての回路を構成することができるのであるが、実際に回路を書くときには基本ゲートだけで書くと煩雑になりがちである。そこで、制御 NOT ゲートの表記を拡張

して、n ビット制御のユニタリゲートを記述できるようにする。即ち、回路図上では、制御 NOT ゲートでは一つだけであった制御線の結線を n 個に増やし、σ_x の代わりに一般のユニタリ作用素 U を書くことで表すことにする。そして、このようなゲートを$\wedge_n(U)$と書く。この表記を使うと、制御 NOT ゲートは$\wedge_1(\sigma_x)$となる。また、$\wedge_0(U)$は U と同じである。$\wedge_n(U)$の入出力ビット数は (U の入出力ビット数) $+ n$ となる。

4 アルゴリズム

以上の議論によって、量子コンピュータを抽象化して回路図で表せることが示せた。よってここからは、量子コンピュータ上のアルゴリズムに関して、回路図を用いながら説明する。

4.1 可逆性

量子コンピュータにおける作用素は、観測を除くと全てユニタリ作用素であり、正則である。即ち、ユニタリ作用素 U の逆作用素 U^{-1} は U^* であり、常に存在する。よって、量子コンピュータ上のアルゴリズムは全て可逆であり、また可逆でなければならない。このことは有利にも不利にも働く。

例えば、古典コンピュータでは最も一般的であるが、量子コンピュータ上では不可逆な操作として、“代入” がある。古典コンピュータ上では

$$a \quad := \quad b$$

という代入は簡単に行える。しかし、一旦 b を代入してしまったら、a にもともと入っていた値は消えてしまうので、この操作は不可逆である。実際、

$$|a\rangle \quad \longmapsto \quad |b\rangle$$

という変換はユニタリ作用素ではないのである。或いはもっと簡単にして

$$|q_0\rangle \quad \longmapsto \quad |0\rangle$$

という変換は、

$$\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$$

という行列で表現されるが、これはユニタリ行列ではない。これから分かるように、単純な代入は量子コンピュータ上ではできないのである。では、値をコピーしたい場合はどのようにするのかというと、

$$|a, 0\rangle \quad \longmapsto \quad |a, a\rangle$$

という変換を考えれば良い。これならば、

$$|a, a\rangle \quad \longmapsto \quad |a, 0\rangle$$

という逆変換が存在するので、量子コンピュータ上で実現可能である。実際に、

$$a \oplus 0 \quad = \quad a$$

であるので、

$$\wedge_1(\sigma_x) \quad : \quad |q_0, 0\rangle \longmapsto |q_0, q_0\rangle$$

となり、これをビット数分だけ繰り返せば良い。しかし、これで全ての問題が解決されたわけではない。それは即ち、このゲートへの入力の半分は必ず 0 を入れなければいけない、ということである。前述の通り、量子コンピュータ上で代入は行えないので、計算途中で強制的にビットを 0 にクリアすることはできない。よって、必ず 0 になっているレジスタ（量子ビットをまとめたものをレジスタと呼ぶ）を用いるか、或いは、まだ使っていないレジスタを用いるかのいずれかである。ただし、未使用のレジスタを使うと、その度に、同時に必要なビット数が増えてしまうため、なるべくならば前者の方法を取るべきである。そして、そのためには、可逆性に影響しないビットは 0 にクリアされた状態で結果が出てくるようなアルゴリズムを設計する必要がある。

逆に、可逆性がメリットとなることは、逆の演算を行う回路が簡単に作れることである。ここで、ある回路 G が $U_1, \cdots, U_n$ というユニタリ行列で表されるゲートから成っていたとすると、

$$\begin{aligned} G &= U_n U_{n-1} \cdots U_1 \\ G^{-1} = G^* &= {U_1}^* {U_2}^* \cdots {U_n}^* \end{aligned}$$

と、逆回路 G^{-1} を求めることができる。

4.2 加算器

量子コンピュータにおける加算器 (adder) は、古典コンピュータにおけるそれとほぼ同じアルゴリズムでできている。ただし、可逆性に起因する、量子コンピュータ独自の部分も含まれている。

まず、1 ビットの加算の下の桁を計算する回路 (SUM) と繰り上がりを計算する回路 (CARRY) を考える。$a_0 + b_0 + c_0 = 2c_1 + s_0$ とすると、

$$\begin{aligned} s_0 &= a_0 \oplus b_0 \oplus c_0 \\ c_1 &= a_0 b_0 + b_0 c_0 + c_0 a_0 \end{aligned}$$

である。s_0 は$\wedge_1(\sigma_x)$を二つ組み合わせれば良いことが分かる。よって、図 5 のようになる。

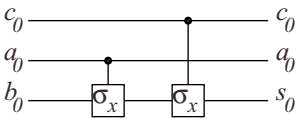

図 5: SUM

CARRY の方は SUM に比べると若干複雑である。そこで、まず、前の桁の繰り上がりを考慮しない加算器 (Half Adder) を考える。

$$a_0 + b_0 \quad = \quad 2{c_1}' + {s_0}'$$

となるので、$|0, b_0, a_0\rangle \longmapsto |c_1, s_0, a_0\rangle$ という変換の真理値表は

0	b_0	a_0	$c_1{}'$	$s_0{}'$	a_0
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	1

である。これから、

$$\begin{aligned} s_0{}' &= a_0 \oplus b_0 \\ c_1{}' &= a_0 b_0 \oplus 0 \end{aligned}$$

ということが分かるので、図 6 のような回路を構成することができる。

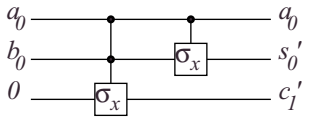

図 6: Half Adder

そして、問題の c_1 は $c_0, s_0{}', c_1{}'$ で決まるので、$|c_1{}', s_0{}', c_0\rangle \longmapsto |c_1, s_0{}', c_0\rangle$ の真理値表を書くと、

$c_1{}'$	$s_0{}'$	c_0	c_1	$s_0{}'$	c_0
0	0	0	0	0	0
0	1	0	0	1	0
1	0	0	1	0	0
0	0	1	0	0	1
0	1	1	1	1	1
1	0	1	1	0	1

となる ($(c_1{}', s_0{}') = (1, 1)$ という組み合わせは半加算器の出力にはない)。ここで、$(s_0{}', c_0) = (1, 1)$ の時のみ $c_1{}' \longmapsto c_1$ でビットが反転していることに注目すると、

$$c_1 = s_0{}' c_0 \oplus c_1{}'$$

ということが分かる。つまり、CARRY の回路は図 7 のようになる。

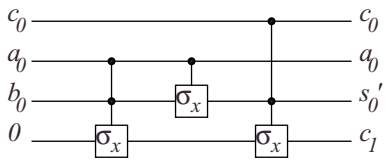

図 7: CARRY

では、SUM(図 5) と CARRY(図 6) を合わせれば全加算器 (Full Adder) が作れるのだろうか？確かに、この二つを合わせて図 8 のようにすれば、これは全加算器の機能を持っている。

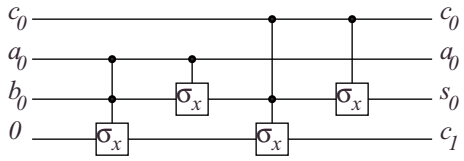

図 8: Full Adder(?)

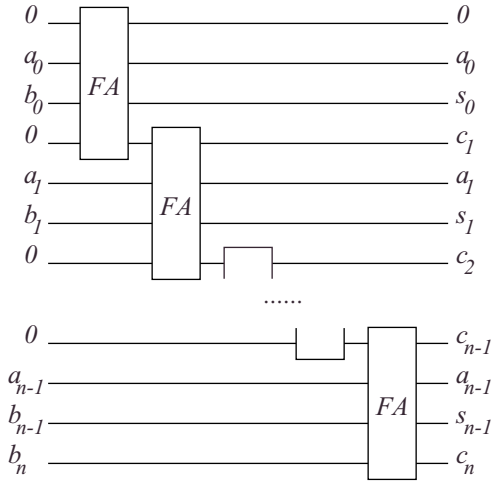

図 9: Full Adder Array

しかし、これをビット数分だけ繋げてみると、ある問題があることに気づく。実際に繋げたものが図 9 である。これによると、入力は a が n ビット、b が $n+1$ ビット、0 が n ビットであるのに対して、出力は a が n ビット、s が $n+1$ ビット、0 が 1 ビット、c が $n-1$ ビットである。しかし、我々が得たいのは s だけである。a が出力されるのは、可逆性のためであるので仕方がないとしても、c の出力は無駄である。即ち、c を使わなくとも、$|a,s\rangle \longmapsto |a,b\rangle$ という逆変換は可能である。そこで、c の出力を打ち消し、0 が出力されるような工夫を考えることにする。

繰り上がり c_1 が使われているのは、その次の桁の FA の中である。また、FA は CARRY と SUM の組み合わせである。つまり、概念的に書くと、

$$(((\cdots)\mathrm{SUM}_2\mathrm{CARRY}_2)\mathrm{SUM}_1\mathrm{CARRY}_1)\mathrm{SUM}_0\mathrm{CARRY}_0$$

という行列の積になっている。SUM_i は a_i, b_i, c_i に依存しているが、
$\mathrm{CARRY}_{i+1}, \mathrm{SUM}_{i+1}, \cdots, \mathrm{CARRY}_{n-1}, \mathrm{SUM}_{n-1}$ によって a_i, b_i, c_i は変化しない。よって、各 SUM_i を後ろに持っていくことができる。

$$\mathrm{SUM}_0(\mathrm{SUM}_1(\mathrm{SUM}_2(\cdots)\mathrm{CARRY}_2)\mathrm{CARRY}_1)\mathrm{CARRY}_0$$

また、同様の理由で、SUM_i の入力は CARRY_i の出力がそのまま保たれている。従って、SUM_i の前に ${\mathrm{CARRY}_i}^{-1}$ を入れれば c_{i+1} は 0 になる。回路図で書くと図 10 となる。あとは、a_i, b_i, c_i を普通の SUM ゲートに通せば s_i を得ることができる。よって、完全な加算器は図 11 のようになる。ADDER_{i+1} の部分には、次の桁以降の加算器が再帰的に入る。また、最も上位の桁 (MSB) 即ち n ビット目に関しては、ADDER_{n+1} の部分は何も入らないので、CARRY と CARRY^{-1} が打ち消しあって SUM だけが残る。その結果、一番下の線は一つもゲートを通らないので、使用ビッ

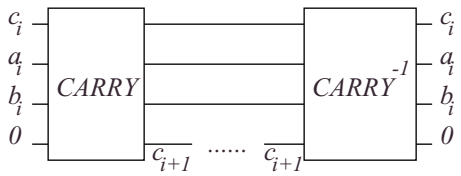

図 10: CARRY and CARRY^{-1}

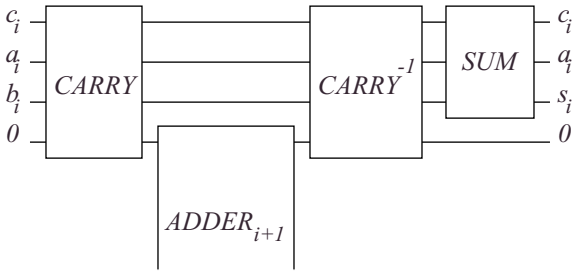

図 11: ADDER$_i$

ト数を一つ減らすことができる。或いは、MSB だけは

$$|b_n, b_{n-1}, a_{n-1}, c_{n-1}\rangle \longmapsto |s_n, s_{n-1}, a_{n-1}, c_{n-1}\rangle$$

という変換を行うようにすることもできる。こうしておいて、さらに $b_n = 0$ としておけば、加算の結果が n ビットに収まらない時に s_n が 1 になるので、キャリーフラグとして用いることができる。このような回路は図 12 として実現できる。

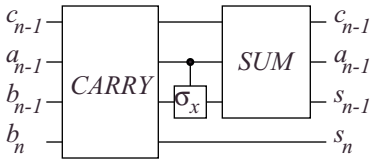

図 12: MSB の ADDER

$c_{n-1}, a_{n-1}, b_{n-1}$ のうち、CARRY ゲートで変化するのは a_{n-1} 制御の b_{n-1} に対する NOT だけであるので、CARRY と SUM の間に$\wedge_1(\sigma_x)$を一つ入れるだけで入力を元の状態に戻すことができる。

以上で n ビット加算器の回路が完成したのであるが、加算器ができるとその逆回路、即ち減算器もできたことになる。$\wedge_m(\sigma_x)^{-1} = \wedge_m(\sigma_x)$ であるので、加算器の回路の左右を反対にしたものが減算器の回路である。減算器は、$|a, s, 0\rangle \longmapsto |a, b, 0\rangle \quad (b = s - a)$ という変換を行い、結果が 0 より小さくなった場合は b_n が 1 になる。

4.3 定数加算器

定数加算器とは、a を入力した時に

$$|a\rangle \longmapsto |a + b\rangle$$

という変換を行う回路である。ただし、ここで使われている b という値は定数であり、回路として組み込まれている値である。また、入出力とも 0 のビットは必要がない限り省略することにする。従って、

$$|a+b\rangle \longmapsto |a\rangle$$

という逆変換が可能である。

実際、定数加算器は加算器を使って実現することができるが、その場合最低限必要なビット数より多くのビットを使うことになってしまうので、別の回路として作ることにする。

基本的には、加算器において各々の入力 a_i を固定し、その状態で要らないゲートを取り除けば、a に対応する定数加算器ができる。定数 0 に対応する CARRY と SUM は、図 13, 14 であり、定数 1 の場合は図 15, 16 である。これらを、CARRY(0), SUM(0); CARRY(1), SUM(1) と呼ぶことにする。

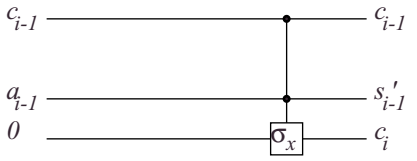

図 13: CARRY(0)

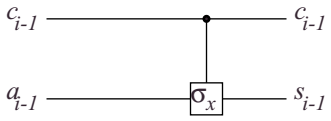

図 14: SUM(0)

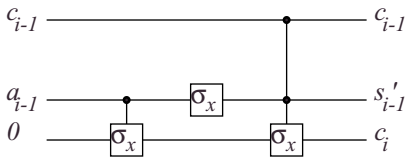

図 15: CARRY(1)

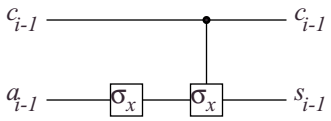

図 16: SUM(1)

後は、加算器の i ビット目の CARRY と SUM を、CARRY(b_{i-1}) と SUM(b_{i-1}) で置き換えれば良い。また、MSB に関しては同様にゲートを整理すると、図 17 の通りになる。

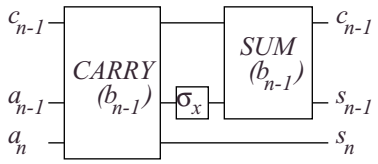

図 17: MSB の定数加算器

加算器の時と同様に、この定数加算器の逆回路は定数減算器になる。

4.4 剰余類環上定数加算器

加算器及び定数加算器は、$\mathbf{Z}$ 上の加算であったが (正確には $\mathbf{Z}/2^n\mathbf{Z}$ 上であるが)、今度は $\mathbf{Z}/m\mathbf{Z}$ 上の定数加算に関して考えてみる。即ち、

$$|a\rangle \longmapsto |a+b \pmod m\rangle$$

という変換を行う回路である。ただし、b, m は回路に組み込まれている値である。ここで、$0 \leq a < m$ であるならば

$$|a + b \pmod m\rangle \longmapsto |a\rangle$$

という逆変換が存在する。そこで、$0 \leq a < m$ という条件に限って話を進めることにする。また、$0 \leq b < m$ としても一般性は失われないので、この範囲の b のみを考える。

a から $a + b \pmod m$ を求めるアルゴリズムを、擬似的なコードで書くと、

```
a := a + b
if ( a ≥ m ) then
    a := a − m
endif
```

となる。一行目は定数加算器そのものである。二行目の判定は、実際に定数減算器で a から m を引き、キャリーフラグを見れば良い。そして、キャリーフラグを制御ビットとして定数減算器に通せば、三行目が実現される。これを元にしてできた回路が図 18 である。ただし、CA, CS はそれぞれ定数加算器と定数減算器を表し、括弧の中に加減算する定数を示している。また、ゲートに “0” と書かれている部分が入出力とも 0 となるビットである。

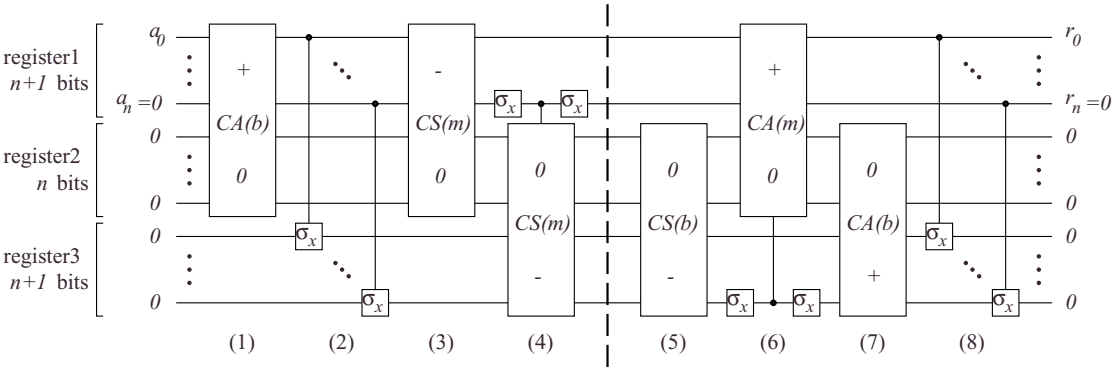

図 18: 剰余類環上定数加算器

図 18 で、(1)〜(8) のゲートによってビットがどのように変化するのかを追っていくと以下のようになる。なお、$|q_1, q_0\rangle$ と書いた場合、q_0 の方が下位ビットであり、従って図の中では上の方の線になる。

(1) 定数加算器でレジスタ 1 に b を足している。

$$|0, 0, a\rangle \quad \longmapsto \quad |0, 0, a + b\rangle$$

(2) レジスタ 1 とレジスタ 3 のそれぞれのビットに$\wedge_1(\sigma_x)$を通している。ここで、レジスタ 3 は 0 に初期化されているので、レジスタ 1 の値がコピーされることになる。

$$\begin{aligned} |0, 0, a + b\rangle \quad &\longmapsto \quad |(a + b) \oplus 0, 0, a + b\rangle \\ &= |a + b, 0, a + b\rangle \end{aligned}$$

(3) 定数減算器でレジスタ 1 から m を引いている。また $a+b<m$ の場合は、その結果、キャリーフラグ (レジスタ 1 の MSB) がセットされる。

$$|a+b, 0, a+b\rangle \quad \longmapsto \quad |a+b, 0, a+b-m\rangle$$

(4) $\wedge_1(CS(m))$にレジスタ 3 を通している。制御ビットはキャリーフラグの反転であるので、(3) で $a+b \geq m$ の場合のみ減算が実行される。よって次のように i), ii) に場合分けをする。

$$|a+b, 0, a+b-m\rangle \quad \longmapsto \quad \begin{cases} |a+b-m, 0, a+b-m\rangle & (a+b \geq m) \ \cdots \text{ i)} \\ |a+b, 0, a+b-m\rangle & (a+b < m) \ \cdots \text{ ii)} \end{cases}$$

この時点で、レジスタ 3 には $a+b \pmod m$ の結果が入っている。しかしこのままでは、0 でないビットが多く出力されてしまう。従ってこれより後は、不必要なビットを 0 にクリアするための回路となる。

(5) レジスタ 3 から b を引いている。

$$\begin{aligned} &\text{i)} && |a+b-m, 0, a+b-m\rangle \ \longmapsto \ |a-m, 0, a+b-m\rangle \\ &\text{ii)} && |a+b, 0, a+b-m\rangle \ \longmapsto \ |a, 0, a+b-m\rangle \end{aligned}$$

(6) (4) と同様に、(5) で (レジスタ 3) $-\, b \geq 0$ の時のみレジスタ 1 に m を加えている。ここで、i) ならば前提条件から、

$$a+b-m-b = a-m \quad < \quad 0$$

となって、実行されることはなく、ii) ならば

$$a+b-b = a \quad \geq \quad 0$$

であるので、必ず実行される。よって、

$$\begin{aligned} &\text{i)} && |a-m, 0, a+b-m\rangle \ \longmapsto \ |a-m, 0, a+b-m\rangle \\ &\text{ii)} && |a, 0, a+b-m\rangle \ \longmapsto \ |a, 0, a+b\rangle \end{aligned}$$

(7) レジスタ 3 に b を加えている。

$$\begin{aligned} &\text{i)} && |a-m, 0, a+b-m\rangle \ \longmapsto \ |a+b-m, 0, a+b-m\rangle \\ &\text{ii)} && |a, 0, a+b\rangle \ \longmapsto \ |a+b, 0, a+b\rangle \end{aligned}$$

(8) レジスタ 1 とレジスタ 3 の XOR をレジスタ 3 に入れているが、どちらの場合においてもレジスタ 1 とレジスタ 3 の値は等しいので、XOR の結果は 0 になる。

$$\begin{aligned} &\text{i)} && |a+b-m, 0, a+b-m\rangle \ \longmapsto \ |0, 0, a+b-m\rangle \\ &\text{ii)} && |a+b, 0, a+b\rangle \ \longmapsto \ |0, 0, a+b\rangle \end{aligned}$$

これで、全体の入出力においてレジスタ 2,3 は 0 となった。

まとめると、

$$|0,0,a\rangle \quad \longmapsto \quad \begin{cases} |0,0,a+b-m\rangle & (a+b \geq m) \\ |0,0,a+b\rangle & (a+b < m) \end{cases} \quad = \quad |0,0,a+b \pmod m\rangle$$

となって、正しく計算できていることが分かる。この回路を、以後、$\mathrm{MCA}(b,m)$ で表すことにする。

ところで、この回路に $a \geq m$ となる a を入れてしまった場合はどうなるのであろうか。$0 \leq a < m$ という条件が使われているのは、(6) であるが、ここで判定を誤ることになり、その結果、レジスタ 3 の値が 0 にならなくなってしまう。さらに、レジスタ 3 の出力は 0 であるものとして他の回路の入力に使われているならば、その回路の結果も狂ってしまう。このように、間違った値を入れるとそれ以降の回路全てに影響が及ぶ可能性がある。

では、$a \geq m$ であっても誤動作の原因とならないようにするには、どのようにすれば良いのであろうか。まず、回路全体の可逆性から、入力の値と出力の値が一対一で対応していなければならない。$0 \leq a < m$ に関しては

$$\mathrm{MCA}(b,m) \quad : \quad \{|a\rangle \,|a \in [0,m)\} \longrightarrow \{|a\rangle \,|a \in [0,m)\}$$

であるので、$a \geq m$ に対して

$$\{|a\rangle \,|a \in [m,2^n)\} \quad \longrightarrow \quad \{|a\rangle \,|a \in [m,2^n)\}$$

という写像を考え、$\mathrm{MCA}(b,m)$ を合わせれば良い。このような写像で最も簡単なのは恒等写像であろう。即ち、

$$\mathrm{MCA}^+(b,m) \quad : \quad \begin{cases} |a\rangle \longmapsto |a+b \pmod m\rangle & (0 \leq a < m) \\ |a\rangle \longmapsto |a\rangle & (a \geq m) \end{cases}$$

と定義すれば、$\mathrm{MCA}^+(b,m)$ は可逆である。これを、擬似コードで書くと、

```
if ( a < m ) then
    a := MCA(b, m)
endif
```

となる。回路図は図 19 である。ただし、$\mathrm{MCA}(b,m)$ のテンポラリビット (入出力が 0 であるビット) は省略した。この回路の動作も、$\mathrm{MCA}(b,m)$ と同様に説明することができる。

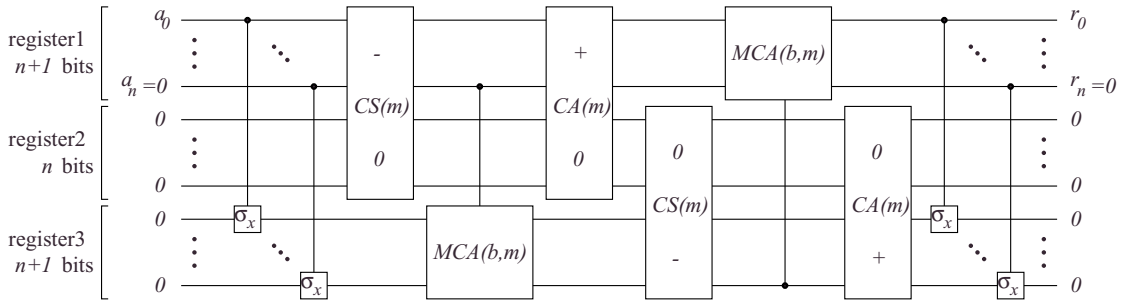

図 19: $\mathrm{MCA}^+(b,m)$

4.5 剰余類環上乗算器

定数加算器ができれば、それを並べることで定数乗算器を作ることができる。即ち、a の二進表現を $a_{n-1}\cdots a_0$ とし、b を定数とすると、

$$\begin{aligned} ab &= b\sum_{i=0}^{n-1} 2^i a_i \\ &= \sum_{i=0}^{n-1} a_i \cdot 2^i b \\ &= \sum_{i:a_i=1} 2^i b \end{aligned}$$

である。これは $\mathbf{Z}/m\mathbf{Z}$ 上でも同様に成り立つ。

$$ab \equiv \sum_{i:a_i=1} 2^i b \pmod{m}$$

さらにこの総和は、

$$\wedge_1(\mathrm{MCA}(2^{n-1}b))(a_{n-1},\cdots\wedge_1(\mathrm{MCA}(2^1 b))(a_1,\wedge_1(\mathrm{MCA}(2^0 b))(a_0,0))\cdots)$$

というように、ゲートで表すことができる。このゲートアレイは

$$|a,0\rangle \longmapsto |a, ab \pmod{m}\rangle$$

という変換を行う。ここで、b と m が互いに素であったとするならば、$bb^{-1}\equiv 1 \pmod{m}$ となる元 b^{-1} が存在する。よって、

$$\begin{aligned} |ab \pmod{m}, 0\rangle &\longmapsto \left|ab \pmod{m}, abb^{-1} \pmod{m}\right\rangle \\ &= |ab \pmod{m}, a\rangle \end{aligned}$$

という変換も可能である。この二つを組み合わせれば、

$$|0,a\rangle \longmapsto |0, ab \pmod{m}\rangle$$

という回路を作ることができる。この回路図は図 20 となる。

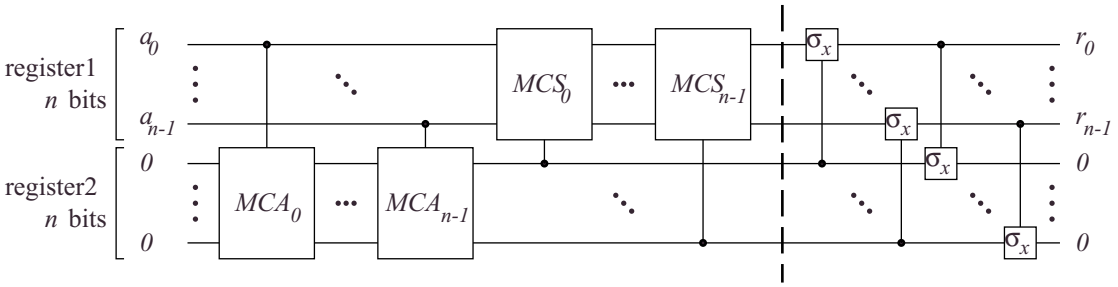

図 20: $\mathrm{MCM}(b,m)$

回路図上のゲートは

$$\begin{aligned} \mathrm{MCA}_i &= \mathrm{MCA}(2^i b, m) \\ \mathrm{MCS}_i &= {\mathrm{MCA}_i}^{-1} \qquad \text{(剰余類環上定数減算)} \end{aligned}$$

となっている。また、破線の位置では、レジスタ 1 が 0 で、レジスタ 2 に $ab \pmod m$ が入っているので、$\wedge_1(\sigma_x)$のゲートアレイでレジスタの内容を入れ替えている。

4.6 剰余類環上累乗器

乗算器を並べることで累乗を計算する回路を作ることができる。これは、乗算器の時とほぼ同様の議論で説明できるが、一つだけ大きく違う点がある。それは即ち、$\mathbf{Z}/m\mathbf{Z}$ 上では累乗の計算は可逆ではない、ということである。何故ならば、x の $\mathrm{mod}m$ での位数を r とすると、

$$\begin{aligned} x^r &\equiv 1 \pmod m \\ \therefore x^{r+k} &\equiv x^k \pmod m \end{aligned}$$

となるからである。よって、累乗を計算する回路は、定数 x, m に対して

$$|0, a\rangle \longmapsto |x^a \pmod m, a\rangle$$

という変換を行う回路になる。乗算器の時と同様に、

$$\begin{aligned} x^a &\equiv x^{\sum_{i=0}^{n-1} 2^i a_i} \pmod m \\ &\equiv \prod_{i=0}^{n-1} x^{2^i a_i} \pmod m \\ &\equiv \prod_{i:a_i=1} x^{2^i} \pmod m \end{aligned}$$

となるので、回路図は図 21 となる。

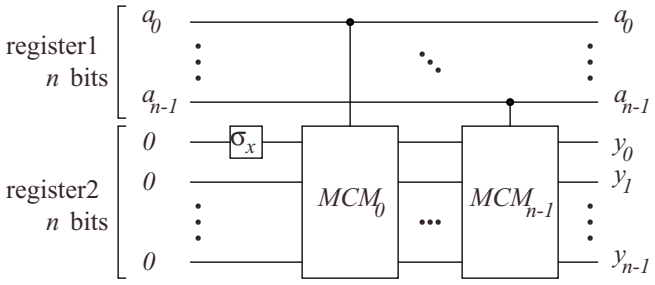

図 21: MCE(x, m)

回路上のゲートは、

$$\mathrm{MCM}_i = \mathrm{MCM}(x^{2^i}, m)$$

となっており、出力は

$$y \equiv x^a \pmod m$$

である。また、MCM で次々と数を掛けていくので、初期値は 1 になっていなければならない。そこで、最初に LSB を反転させて 1 にしている。

4.7 離散フーリエ変換 (DFT)

一般の離散フーリエ変換は、時間成分 $|x\rangle = {}^t(x_0, \cdots, x_N)$ から周波数成分 $|X\rangle = {}^t(X_0, \cdots, X_N)$ への変換である。即ち、

$$\begin{aligned} w &= e^{2\pi i/N} \\ D &= \frac{1}{\sqrt{N}} \sum_{j_1, j_2=0}^{N-1} w^{j_1 j_2} |j_1\rangle \langle j_2| \end{aligned}$$

という D によって、

$$|X\rangle = D|x\rangle$$

と変換される。ここで、$N = 2^m$ とし、行列 D の行ベクトルを一定の規則で入れ替えた行列 A を考える。

$$\begin{aligned} A &= A_0(\theta_0) \\ \theta_0 &= e^{2\pi i} = 1 \\ A_k(\theta_k) &= \frac{1}{\sqrt{2}} \begin{pmatrix} A_{k+1}(\sqrt{\theta_k}) & \sqrt{\theta_k} A_{k+1}(\sqrt{\theta_k}) \\ A_{k+1}(-\sqrt{\theta_k}) & -\sqrt{\theta_k} A_{k+1}(-\sqrt{\theta_k}) \end{pmatrix} \\ &= \frac{1}{\sqrt{2}} \sum_{j_1, j_2=0}^{1} \left(e^{i\pi j_1} \sqrt{\theta_k}\right)^{j_2} \left(|j_1\rangle \langle j_2| \otimes A_{k+1}\left(e^{i\pi j_1} \sqrt{\theta_k}\right)\right) \\ A_m(\theta_m) &= 1 \end{aligned}$$

これが意味することは、$A_k(\theta_k)$ を

$$A_k(\theta_k) = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix}$$

と四等分すると、B_2 は B_1 の各成分の位相を $\arg(\theta_k)/2$ だけ進めたものとなっており、また、B_4 は B_3 の各成分の位相を $\arg(\theta_k)/2 + \pi$ だけ進めたものとなっている、ということである。ただし、$0 \leq \arg(\theta_k) < 2\pi$ とする。また、$\arg(\theta_k)$ は A_k 全体の位相差を表す。よって、中間点での位相は、

$$\arg(\pm\sqrt{\theta_k})$$

であり、二つずつ対になっているので、必ずこのような分割を行うことができる。

ところで、

$$A_k(\theta_k) = \begin{pmatrix} A_{k+1}(\sqrt{\theta_k}) & 0 \\ 0 & A_{k+1}(-\sqrt{\theta_k}) \end{pmatrix} \left(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes I\right) \left(\begin{pmatrix} 1 & 0 \\ 0 & \sqrt{\theta_k} \end{pmatrix} \otimes I\right)$$

であり、$|b_k \cdots b_m\rangle$ を基底とすると、

$$\begin{aligned} & \begin{pmatrix} A_{k+1}(\sqrt{\theta_k}) & 0 \\ 0 & A_{k+1}(-\sqrt{\theta_k}) \end{pmatrix} |b_k \cdots b_m\rangle \\ &= \begin{cases} |b_k\rangle \otimes (A_{k+1}(\sqrt{\theta_k}) |b_{k+1} \cdots b_m\rangle) & (b_k = 0) \\ |b_k\rangle \otimes (A_{k+1}(-\sqrt{\theta_k}) |b_{k+1} \cdots b_m\rangle) & (b_k = 1) \end{cases} \\ &= |b_k\rangle \otimes (A_{k+1}(e^{i\pi b_k} \sqrt{\theta_k}) |b_{k+1} \cdots b_m\rangle) \end{aligned}$$

である。よって、基底 $|b_k \cdots b_m\rangle$ を定めると、A_{k+1} のパラメータが

$$\theta_{k+1} \quad = \quad e^{i\pi b_k}\sqrt{\theta_k} \tag{12}$$

と決定される。また、

$$\theta_0 \quad = \quad 1 \tag{13}$$

である。(12), (13) の漸化式は

$$\begin{cases} \log\theta_{k+1} = i\pi b_k + \frac{1}{2}\log\theta_k \\ \log\theta_0 = 0 \end{cases}$$

となるので、これを解くと、

$$\begin{aligned}
\log\theta_k &= i\pi\sum_{j=0}^{k-1}\left(\frac{1}{2}\right)^{k-1-j} b_j \\
&= i\pi\sum_{j=0}^{k-1} 2^{-(k-1-j)} b_j \\
&= i\pi\cdot 2^{-k+1}\sum_{j=0}^{k-1} 2^j b_j \\
\theta_k &= \exp\left(i\pi\cdot 2^{-k+1}\sum_{j=0}^{k-1} 2^j b_j\right) \\
\sqrt{\theta_k} &= \exp\left(i\pi\cdot 2^{-k}\sum_{j=0}^{k-1} 2^j b_j\right) \\
&= \exp\left(i\pi\cdot 2^{-k}\sum_{j:b_j=1} 2^j\right) \\
&= \prod_{j:b_j=1}\exp\left(i\pi\cdot 2^{j-k}\right)
\end{aligned}$$

である。従って、

$$\begin{pmatrix} 1 & 0 \\ 0 & \sqrt{\theta_k} \end{pmatrix} \quad = \quad \prod_{j:b_j=1}\begin{pmatrix} 1 & 0 \\ 0 & \exp\left(i\pi\cdot 2^{j-k}\right) \end{pmatrix}$$

なので、

$$\begin{aligned}
R &= \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \\
S_{jk} &= \begin{pmatrix} 1 & 0 \\ 0 & \exp\left(i\pi\cdot 2^{j-k}\right) \end{pmatrix}
\end{aligned}$$

とすると、b_j 制御の$\wedge_1(S_{jk})$を並べ、その後に R を置くことで、回路として構成できる。

A は D の行ベクトルを入れ替えたものであったから、

$$A\,|x\rangle$$

の結果は、$|X\rangle$ の D の行ベクトルに対応する成分を入れ替えたものになっている。そこで、A の第 $b_0 \cdots b_{m-1}$ 行がどうなっているのかを見ると、

$$
\begin{aligned}
&\langle b_k \cdots b_{m-1}| A_k(\theta_k) \\
&= \langle b_k \cdots b_{m-1}| \cdot \frac{1}{\sqrt{2}} \sum_{j_1,j_2=0}^{1} \left(e^{i\pi j_1}\sqrt{\theta_k}\right)^{j_2} \left(|j_1\rangle\langle j_2| \otimes A_{k+1}\left(e^{i\pi j_1}\sqrt{\theta_k}\right)\right) \\
&= \frac{1}{\sqrt{2}} \sum_{j_1,j_2=0}^{1} \left(e^{i\pi j_1}\sqrt{\theta_k}\right)^{j_2} (\langle b_k|j_1\rangle\langle j_2|) \otimes \left(\langle b_{k+1} \cdots b_{m-1}| A_{k+1}\left(e^{i\pi j_1}\sqrt{\theta_k}\right)\right) \\
&= \frac{1}{\sqrt{2}} \sum_{j_2=0}^{1} \left(e^{i\pi b_k}\sqrt{\theta_k}\right)^{j_2} \langle j_2| \otimes \left(\langle b_{k+1} \cdots b_{m-1}| A_{k+1}\left(e^{i\pi b_k}\sqrt{\theta_k}\right)\right) \\
&= \frac{1}{\sqrt{2}} \sum_{j_2=0}^{1} {\theta_{k+1}}^{j_2} \langle j_2| \otimes (\langle b_{k+1} \cdots b_{m-1}| A_{k+1}(\theta_{k+1}))
\end{aligned}
$$

であるので、θ_k は先ほどと同じになり、

$$
\begin{aligned}
&\langle b_0 \cdots b_{m-1}| A \\
&= \langle b_0 \cdots b_{m-1}| A_0(\theta_0) \\
&= \frac{1}{\sqrt{2}} \sum_{j_0=0}^{1} {\theta_1}^{j_0} \langle j_0| \otimes \frac{1}{\sqrt{2}} \sum_{j_1=0}^{1} {\theta_2}^{j_1} \langle j_1| \otimes \cdots \otimes \frac{1}{\sqrt{2}} \sum_{j_{m-1}=0}^{1} {\theta_m}^{j_{m-1}} \langle j_{m-1}| \\
&= \left(\frac{1}{\sqrt{2}}\right)^m \sum_{j_0,\cdots,j_{m-1}=0}^{1} \left(\prod_{l=0}^{m-1} {\theta_{l+1}}^{j_l}\right) \langle j_0 \cdots j_{m-1}|
\end{aligned}
$$

となっている。この第二成分、即ち $\langle 0 \cdots 01|$ の成分は、

$$
\begin{aligned}
&\langle b_0 \cdots b_{m-1}| A |0 \cdots 01\rangle \\
&= \left(\frac{1}{\sqrt{2}}\right)^m \theta_m \\
&= \left(\frac{1}{\sqrt{2}}\right)^m \exp\left(i\pi \cdot 2^{-(m-1)} \sum_{j=0}^{m-1} 2^j b_j\right)
\end{aligned}
$$

である。一方、D については、

$$
\begin{aligned}
&\langle b_{m-1} \cdots b_0| D |0 \cdots 01\rangle \\
&= \langle b_{m-1} \cdots b_0| \left(\frac{1}{\sqrt{N}} \sum_{j_1,j_2=0}^{N-1} w^{j_1 j_2} |j_1\rangle\langle j_2|\right) |0 \cdots 01\rangle \\
&= \left(\frac{1}{\sqrt{2}}\right)^m \sum_{j_1=0}^{N-1} w^{j_1} \langle b_{m-1} \cdots b_0|j_1\rangle \\
&= \left(\frac{1}{\sqrt{2}}\right)^m \sum_{j_{m-1},\cdots,j_0=0}^{1} \exp\left(2\pi i \cdot \frac{1}{2^m} \sum_{l=0}^{m-1} 2^l j_l\right) \langle b_{m-1} \cdots b_0|j_{m-1} \cdots j_0\rangle \\
&= \left(\frac{1}{\sqrt{2}}\right)^m \exp\left(i\pi \cdot 2^{-(m-1)} \sum_{l=0}^{m-1} 2^l b_l\right)
\end{aligned}
$$

であるので、

$$\langle b_0 \cdots b_{m-1}| A |0 \cdots 01\rangle \quad = \quad \langle b_{m-1} \cdots b_0| D |0 \cdots 01\rangle$$

となる。従って、A の第 $b_0 \cdots b_{m-1}$ 行と D の第 $b_{m-1} \cdots b_0$ 行が等しいことになる。即ち、

$$C \quad : \quad |b_0 \cdots b_{m-1}\rangle \longmapsto |b_{m-1} \cdots b_0\rangle$$

とすると、

$$D \quad = \quad CA$$

ということである。C はビットオーダを逆順にする変換である。

以上をまとめると図 22 の回路ができる。

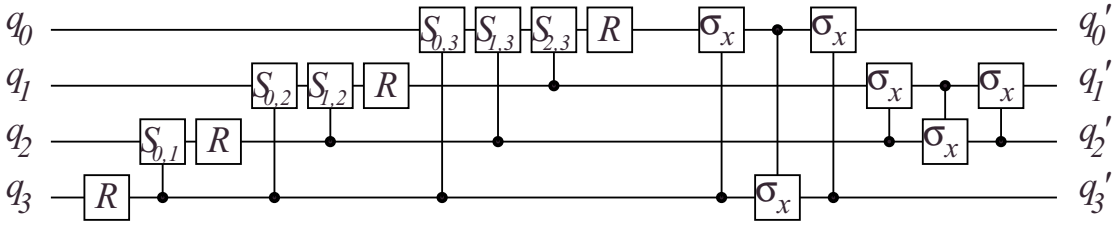

図 22: DFT(4bits)

4.8 位数計算

図 23 の回路を使って、定数 x, m に対して $\bmod m$ における x の位数を求めることができる。ただし、M というゲートは、そのビットを観測することを表す。

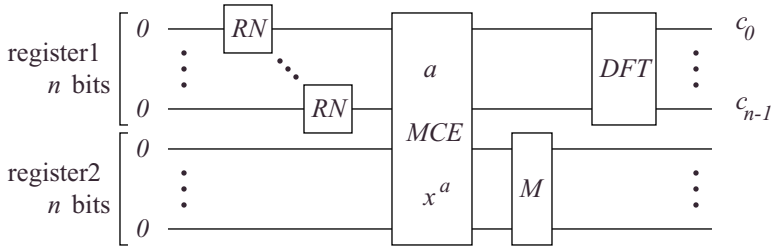

図 23: 位数計算用回路

まず、$\sqrt{\mathrm{NOT}}$によってレジスタ 1 の確率振動を一様に分布させる。つまり、

$$\frac{1}{\sqrt{2^n}} \sum_{a=0}^{2^n-1} |0, a\rangle$$

という状態になる。次に、MCE(x, m) によって、$x^a \pmod m$ を計算する。

$$\frac{1}{\sqrt{2^n}} \sum_{a=0}^{2^n-1} |x^a (\mathrm{mod} m), a\rangle$$

この状態でレジスタ 2 を観測すると、その値が一つに決まる。観測後の値を $x^k \pmod m$ とすると、量子コンピュータの状態は

$$\frac{1}{A}\sum_{a:x^a\equiv x^k}|x^a(\mathrm{mod}m),a\rangle\,,\qquad A=\left|\sum_{a:x^a\equiv x^k}|x^a(\mathrm{mod}m),a\rangle\right|$$

となる。ここで、求めるべき x の位数を r とすると、

$$\begin{aligned}&x^a\equiv x^k \pmod m\\ \Longleftrightarrow\ &a\equiv k \pmod r\end{aligned}$$

であるので、$0\le k<r$ とし、a を k,r で表すと、

$$a \;=\; br+k \qquad (b\in\mathbf{Z})$$

となる。即ち、k の値に依らず、レジスタ 1 の成分は周期 r で 0 以外の値が存在している。よって、DFT によって周波数成分に変換すると、周期 r つまり周波数 $2^n/r$ とその整数倍の成分の確率振動が大きくなる。よって、この状態でレジスタ 1 を観測した結果を c とすると、

$$\frac{c}{2^n}\approx\frac{d}{r}\qquad(d\in\mathbf{Z})$$

である。そこで、$\frac{c}{2^n}$ を連分数展開し、$0<r<m$ という条件の元で最も良い近似となるものを見つければ、その分母が r である。

[1] に依ると、ある定数 δ が存在して、ランダムな値の x に対してこのようにして r が求められる確率 p は、

$$p>\frac{\delta}{\log\log r}$$

と下から押さえられている。ここで、確率変数 N を、r が見つかるまでの回数とする。すると、N の平均 $E(N)$ は、幾何分布 $G(\delta/\log\log r)$ の平均で上から押さえられることになる。よって、

$$\begin{aligned}E(N) \;&<\; E\left(G\left(\frac{\delta}{\log\log r}\right)\right)\\ &=\;\frac{1-\frac{\delta}{\log\log r}}{\frac{\delta}{\log\log r}}\\ &=\;\frac{\log\log r-\delta}{\delta}\\ &=\;O(\log\log r)\end{aligned}$$

となる。

4.9 因数分解

まず、因数分解したい数を m とし、ランダムな数 $x(<m)$ を選ぶ。もし、

$$\gcd(m,x)>1$$

ならば、$\gcd(m,x)$ が m の因数である。そうでないのならば、前述のアルゴリズムで、mod$m$ での x の位数 r を求められる。ここで、r が偶数であったとすると、

$$\begin{aligned} x^r &\equiv 1 \pmod m \\ x^r - 1 &\equiv 0 \pmod m \\ \therefore (x^{r/2}-1)(x^{r/2}+1) &\equiv 0 \pmod m \end{aligned}$$

となっている。また、位数の最小性から、$x^{r/2} \not\equiv 1 \pmod m$ である。そして、

$$x^{r/2} \not\equiv -1 \pmod m$$

と仮定すると、

$$\begin{aligned} (x^{r/2}-1)(x^{r/2}+1) &\equiv 0 \pmod m \\ x^{r/2}-1 &\not\equiv 0 \pmod m \\ x^{r/2}+1 &\not\equiv 0 \pmod m \end{aligned}$$

であるので、適当な整数 p,q が存在して、

$$\begin{aligned} p,q &> 1 \\ m &= pq \\ x^{r/2}-1 &\equiv 0 \pmod p \\ x^{r/2}+1 &\equiv 0 \pmod q \end{aligned}$$

である。従って、

$$1 < \gcd(m, x^{r/2}-1) < m$$

となり、$\gcd(m, x^{r/2}-1)$ が m の因数となっている。

以上から、

1. r が偶数である。
2. $x^{r/2} \not\equiv -1 \pmod m$ である。

の二つが成り立てば、m の因数を求めることができる。[1] によれば、ランダムな x に対してこれらの条件が成り立つ確率は、m の偶素因数の数を k とすると、

$$1 - \left(\frac{1}{2}\right)^{k-1}$$

より大きい。よって、$k > 1$ ならば、平均で定数回繰り返せば成功する。また、$k = 0$ ならば、m は 2 の累乗数である。そして、$k = 1$ で m が 2 を因数として持たない場合は、m は累乗数であり、$\log m$ の多項式時間で因数を求められる。

計算量　では、m の因数を求めるのにかかる計算量を調べてみる。まず、CA, CS の計算量、即ち基本ゲートの数は $O(\log m)$ である。また、MCA, MCS は、3 個の CA、3 個の CS、$\log m$ 個の $\wedge_1(\sigma_x)$、4 個の σ_x で構成されているので、その計算量は同じく $O(\log m)$ である。そして、MCM

は、$\log m$ 個の$\wedge_1$(MCA)と$\wedge_1$(MCS)、$2\log m$ 個の$\wedge_1(\sigma_x)$から成るので、計算量は $O((\log m)^2)$ である。同様に、MCE は、$\log m$ 個の MCM と 1 個の σ_x からできているので、計算量は $O((\log m)^3)$ となる。さらに、DFT は、$\log m$ 個の R と $\log m(\log m - 1)/2$ 個の S_{jk} で構成されているので、計算量は $O((\log m)^2)$ である。

以上から、位数計算回路は $\log m$ 個の$\sqrt{\text{NOT}}$、1 個の MCE、1 個の DFT でできているので、その計算量は $O((\log m)^3)$ となる。また、位数計算回路を平均 $O(\log\log r)$ 回繰り返すと、r が求まるので、その全体の計算量は $O((\log m)^3 \log\log m)$ である。位数 r から m の因数を求めるのは、平均で定数回でできるので、因数分解の計算量は $O((\log m)^3 \log\log m)$ となり、$\log m$ 即ち m のビット数の多項式時間で実行できる。

4.10 RSA 暗号の解読

RSA 暗号と因数分解の関係について説明する。

まず、RSA 暗号のアルゴリズムであるが、任意の相異なる素数 p, q, e を定めた時、

$$\begin{aligned} L &= (p-1)(q-1) \\ n &= pq \end{aligned}$$

とすると、

$$\begin{aligned} \gcd(x, n) \quad &\Rightarrow \quad x^L \equiv 1 \pmod{m} \\ \exists d, y \quad &\qquad ed + Ly = 1 \end{aligned}$$

である。よって、

$$\begin{aligned} x &= x^{ed+Ly} \\ &= x^{ed} x^{Ly} \\ &= x^{ed} (x^L)^y \\ &\equiv x^{ed} 1^y \pmod{n} \\ &= x^{ed} \end{aligned}$$

であるので、x を平文とすると、

$$x \quad \xrightarrow{\text{暗号化}} \quad x^e \pmod{n} \quad \xrightarrow{\text{復号化}} \quad (x^e)^d \equiv x \pmod{n}$$

とできる。そして、ここで使われた情報のうち、暗号化に必要な e, n だけが公開される。また、d が秘密鍵となる。

従って、RSA 暗号の解読とは、e, n から秘密鍵 d を求めることである。ここで、もし n を因数分解することができるならば、その因数 p, q が求まることになり、そして p, q, e が分かれば他の値もわかる。即ち、n が因数分解できると RSA 暗号は破られるのである。

現在実現されている古典コンピュータでは、大きな数の因数分解に関して効率的なアルゴリズムが発見されていない。よって、RSA 暗号の信頼性は計算複雑性から来ている。しかし、量子コンピュータが実現されれば、これまで述べてきた方法によって効率的に因数分解が可能であるので、RSA 暗号の信頼性が落ちることになる。

5 シミュレーション

以上までで量子コンピュータ上で動くアルゴリズムを論じてきたが、量子コンピュータがまだ実用化されていない現段階では、実際にアルゴリズムを動かしてみることは不可能である。そこで、量子コンピュータの動作を古典コンピュータ上でシミュレートし、その動作について理解を深めるために、*Mathematica* 上でシミュレータを作成することにする。

5.1 簡易リファレンス

5.1.1 基底ベクトル

基底ベクトル $|i\rangle = |i_{n-1} \cdots i_0\rangle$ $(i_j \in \{0, 1\})$ を

```
ket[i0,i1,...]
```

と表す。$i_0, \cdots, i_{n-1}$ の順番が逆になって、リトルエンディアン (下位ビットが先行) になっていることに注意する。

5.1.2 ゲート

状態、即ち基底ベクトルの線形結合にゲートを適用することで状態を変化させる。ゲートとしては、

- CheckGate
- MatrixGate
- QNot
- MeasurementGate
- BitOrderArrangementGate

が用意されており、またゲートアダプタとして、

- GateArray
- SelectedGate
- ControlledGate
- InversedGate

がある。また、良く利用される特定のゲートもあらかじめ定義されている。

- CNot
- RootNot

ゲート

CheckGate
これは本来の量子ゲートではなく、デバッグ用のゲートである。このゲートを通る時に、量子コンピュータの状態が通知関数 (指定しなければ `Print`) に通知される。

MatrixGate
一般のユニタリゲートである。行列の次数 2^n に従って、下位から n ビットが操作の対象となる。

QNot
NOT ゲートである。`Not` と区別するために先頭に “Q(uantum)” をつけている。

MeasurementGate
観測ゲートである。ビット数 n を指定し、下位 n ビットを観測する。このゲートは後述のゲートアダプタ ControlledGate, InversedGate と組み合わせて使うことはできない。

BitOrderArrangementGate
基底ベクトルのビット順を並べ替えるゲートである。SelectedGate の中で用いられているが、直接使う必要はほとんどない。

CNot
制御 NOT ゲートである。制御ビット数 n を指定し、下位 n ビットで $n+1$ ビット目の NOT を制御する。

RootNot
$\sqrt{\mathrm{NOT}}$ゲートである。下位 1 ビットに対して作用する。

ゲートアダプタ

GateArray
複数のゲートをまとめるためのゲートアダプタである。GateArray 自体も一つのゲートであるかのように振舞う。

SelectedGate
ゲートへの入力ビットを選択するためのゲートアダプタである。実際には指定された順に下位ビットから並べ、指定されなかったビットはそれより上位に移動してから、ゲートに通す。また、ゲートに通した後は元のビット順に並べ替えなおす。

ControlledGate
一般の制御付ゲートを作るためのゲートアダプタである。制御ビット数 n を指定し、下位 n ビットで制御、$n+1$ ビット目以降にゲートを適用、となる。

InversedGate
指定されたゲートの逆ゲートとして働くゲートアダプタである。MeasurementGate と CheckGate 以外の全てのゲートに対して用いることができる。

5.2 ソース

5.2.1 シミュレータ

```
Clear[SetLinearity];
SetAttributes[SetLinearity,HoldAll];
SetLinearity[f_,ff_]:=(
  f[a_+b_]:=Expand[ff[a]+ff[b]];
  f[a_?NumericQ*b_]:=Expand[a*ff[b]];
  f[0]:=0;
 );
SetLinearity[f_]:=SetLinearity[f,f];

Clear[SetBilinearity];
SetBilinearity[f_]:=(
  f[x1_+x2_,y_]:=f[x1,y]+f[x2,y];
  f[x_,y1_+y2_]:=f[x,y1]+f[x,y2];
  f[a_?NumericQ*x_,y_]:=a*f[x,y];
  f[x_,a_?NumericQ*y_]:=a*f[x,y];
  f[0,x_]:=0;
  f[x_,0]:=0;
 );

Unprotect[Dot];
Clear[Dot];
SetBilinearity[Dot];
Dot[bra[x__],ket[x__]]:=1;
Dot[bra[x__],ket[y__]]:=0;
Dot[x_,y_?NumericQ]:=y*x;
Dot[x_?NumericQ,y_]:=x*y;
Protect[Dot];

Clear[CircleTimes];
SetAttributes[CircleTimes,Flat];
SetBilinearity[CircleTimes];
CircleTimes[ket[x___],ket[y___]]:=ket[x,y];
CircleTimes[bra[x___],bra[y___]]:=bra[x,y];
CircleTimes[Dot[x1_ket,x2_bra],Dot[y1_ket,y2_bra]]:=
  Dot[CircleTimes[x1,y1],CircleTimes[x2,y2]];

Clear[ComplexConjugate];
ComplexConjugate[x_?NumericQ]:=Conjugate[x];
ComplexConjugate[bra[x__]]:=ket[x];
ComplexConjugate[ket[x__]]:=bra[x];
```

```
ComplexConjugate[x_+y_]:=ComplexConjugate[x]+ComplexConjugate[y];
ComplexConjugate[Dot[x_,y_]]:=Dot[ComplexConjugate[y],ComplexConjugate[x]];
ComplexConjugate[x_*y_]:=ComplexConjugate[x]*ComplexConjugate[y];

Clear[SquareMagnitude,Magnitude];
SquareMagnitude[x_?NumericQ]:=x*Conjugate[x];
SquareMagnitude[v_]:=
  Apply[Plus,Map[SquareMagnitude,Coefficient[v,Variables[v]]]];
Magnitude[x_]:=Sqrt[SquareMagnitude[x]];

Clear[Normalize];
Normalize[v_]:=v/Magnitude[v];

Clear[GetDimension];
GetDimension[x_ket]:=Length[x];
GetDimension[x_bra]:=Length[x];
GetDimension[Dot[x_ket,y_bra]]:=GetDimension[x];
GetDimension[x_+y_]:=GetDimension[x];
GetDimension[a_?NumericQ*x_]:=GetDimension[x];

Clear[SplitVector];
SplitVector[x_ket,pos_Integer]:={Part[x,Range[1,pos]],
    Part[x,Range[pos+1,GetDimension[x]]]};
SplitVector[x_ket,len_Integer,rest__Integer]:=
  MapAt[SplitVector[#,rest]&,SplitVector[x,len],2]//Flatten;

Clear[GetIndex];
GetIndex[ket[x_]]:=x;
GetIndex[bra[x_]]:=x;
GetIndex[ket[x_,y__]]:=x+GetIndex[ket[y]]*2;
GetIndex[bra[x_,y__]]:=x+GetIndex[bra[y]]*2;

Clear[StandardMatrix];
SetLinearity[StandardMatrix];
StandardMatrix[Dot[x_ket,y_bra]]:=Module[{t},
  t=Table[0,{i,1,2^GetDimension[x]},{j,1,2^GetDimension[y]}];
  ReplacePart[t,1,{GetIndex[x]+1,GetIndex[y]+1}]
 ];

Clear[GetIdentityMatrix];
GetIdentityMatrix[len_Integer]:=
  Apply[Plus,
    Outer[ket[##].bra[##]&,Apply[Sequence,Table[{0,1},{len}]]]//Flatten];
```

```
Clear[IdentityQ];
IdentityQ[matrix_]:=(matrix-GetIdentityMatrix[GetDimension[matrix]]===0);

Clear[UnitaryQ];
UnitaryQ[matrix_]:=IdentityQ[matrix.ComplexConjugate[matrix]];

Clear[HermitianQ];
HermitianQ[matrix_]:=(matrix-ComplexConjugate[matrix]===0);

Clear[IdentityGate];
IdentityGate[x_]:=x;
InversedGate[IdentityGate]^:=IdentityGate;

Clear[GateArray];
GateArray[gates___][r_]:=Reverse[Composition[gates]][r];
InversedGate[ga_GateArray]^:=Map[InversedGate,Reverse[ga]];

Clear[CheckGate];
CheckGate[notify_:Print][r_]:=(notify[r];r);

Clear[MatrixGate];
SetLinearity[MatrixGate[x_],MatrixGate[x]];
MatrixGate[matrix_][r_ket]:=
  Apply[CircleTimes,MapAt[matrix.#&,SplitVector[r,GetDimension[matrix]],1]];
InversedGate[MatrixGate[matrix_]]^:=MatrixGate[ComplexConjugate[matrix]];

Clear[QNot];
SetLinearity[QNot];
QNot[ket[0,q___]]:=ket[1,q];
QNot[ket[1,q___]]:=ket[0,q];
InversedGate[QNot]^:=QNot;

Clear[CNot];
CNot[n_Integer]:=ControlledGate[QNot,n];

Clear[RootNot];
RootNot:=MatrixGate[
    1/Sqrt[2]*(ket[0].bra[0]+ket[1].bra[0]-ket[0].bra[1]+ket[1].bra[1])];

SeedRandom[];
Clear[MeasurementGate];
MeasurementGate[qlist:{{__ket}..}][r_]:=Module[{prob,s,i,rnd},
```

```
  prob=Map[Coefficient[r,#]&,qlist];
  prob=Map[SquareMagnitude,prob,{2}];
  prob=Apply[Plus,prob,{1}];
  rnd=Random[];
  For[s=0;i=0,s<=rnd&&i<Length[prob],,s+=prob[[++i]]];
  Apply[Plus,Coefficient[r,qlist[[i]]]*qlist[[i]]/Sqrt[prob[[i]]]//Expand
 ];
MeasurementGate[len_Integer][r_]:=
  Module[{reg,rnd,s,i,vars,vars2,coeffs,prob,pos},
  reg=Expand[r];
  vars=Variables[reg];
  vars2=Map[SplitVector[#,len][[1]]&,vars];
  coeffs=Coefficient[reg,vars];
  prob=Map[SquareMagnitude,coeffs];
  rnd=Random[];
  For[s=0;i=0,s<=rnd&&i<Length[vars],,s+=prob[[++i]]];
  pos=Position[vars2,vars2[[i]],{1}]//Flatten;
  coeffs=coeffs[[pos]];
  prob=Apply[Plus,prob[[pos]]];
  vars=vars[[pos]];
  Apply[Plus,coeffs*vars/Sqrt[prob]]
 ];

Clear[BitOrderArrangementGate];
SetLinearity[BitOrderArrangementGate[x:{___Integer}],
  BitOrderArrangementGate[x]];
BitOrderArrangementGate[x:{___Integer}][r_ket]:=Part[r,x];
InversedGate[BitOrderArrangementGate[x:{___Integer}]]^:=Module[{z},
  z=Table[0,{Length[x]}];
  Do[z[[x[[i]]]]=i,{i,Length[x]}];
  BitOrderArrangementGate[z]
 ];

Clear[SelectedGate];
SelectedGate[gate_,x_][0]:=0;
SelectedGate[gate_,x:{___Integer}][r_]:=Module[{y},
  y=Join[x,Complement[Table[i,{i,GetDimension[r]}],x]];
  r//BitOrderArrangementGate[y]//gate//
      InversedGate[BitOrderArrangementGate[y]]
];
InversedGate[SelectedGate[gate_,x_]]^:=SelectedGate[InversedGate[gate],x];

Clear[ControlledGate];
```

```
ControlledGate[gate_,n_Integer][0]:=0;
ControlledGate[gate_,0]:=gate;
ControlledGate[gate_,n_Integer][r_]:=Module[{vars,vars2,coeffs,pos,pos2},
  vars=Variables[r];
  vars2=Map[SplitVector[#,n][[1]]&,vars];
  coeffs=Coefficient[r,vars];
  pos=Position[vars2,Apply[ket,Table[1,{n}]],{1}]//Flatten;
  pos2=Complement[Range[1,Length[vars]],pos];
  If[pos=={},0,
        SelectedGate[gate,Range[n+1,GetDimension[r]]][
          Apply[Plus,coeffs[[pos]]*vars[[pos]]]]]
    +Apply[Plus,coeffs[[pos2]]*vars[[pos2]]]
 ];
InversedGate[ControlledGate[gate_,n_Integer]]^:=
  ControlledGate[InversedGate[gate],n];

Clear[GateToMatrix];
GateToMatrix[gate_,nqb_]:=
  Apply[Plus,
      Outer[Dot[gate[ket[##]],bra[##]]&,Apply[Sequence,Table[{0,1},{nqb}]],{
        0,nqb-1}]//Expand;
```

5.2.2 アルゴリズム

```
Clear[carry];
carry:=GateArray[
  SelectedGate[CNot[2],{2,3,4}],
  SelectedGate[CNot[1],{2,3}],
  SelectedGate[CNot[2],{1,3,4}]
 ];

Clear[sum];
sum:=GateArray[
  SelectedGate[CNot[1],{1,3}],
  SelectedGate[CNot[1],{2,3}]
 ];

Clear[Adder];
Adder[1]:=Module[{a,b,c},
  a[i_]:=i;
  b[i_]:=i+1;
  c[i_]:=i+2+1;
  GateArray[
```

```
   SelectedGate[carry,{c[1],a[1],b[1],b[2]}],
   SelectedGate[CNot[1],{a[1],b[1]}],
   SelectedGate[sum,{c[1],a[1],b[1]}]
  ]
 ];
Adder[n_Integer]:=Module[{a,b,c},
  a[i_]:=i;
  b[i_]:=i+n;
  c[i_]:=i+2*n+1;
  GateArray[
   SelectedGate[carry,{c[1],a[1],b[1],c[2]}],
   SelectedGate[Adder[n-1],
        Join[Table[a[i],{i,2,n}],Table[b[i],{i,2,n+1}],Table[c[i],{i,2,n}]]],
   SelectedGate[InversedGate[carry],{c[1],a[1],b[1],c[2]}],
   SelectedGate[sum,{c[1],a[1],b[1]}]
  ]
 ];

Clear[Subtractor];
Subtractor[n_Integer]:=InversedGate[Adder[n]];

Clear[ConstCarry];
ConstCarry[0]:=CNot[2];
ConstCarry[1]:=GateArray[
  SelectedGate[CNot[1],{2,3}],
  SelectedGate[QNot,{2}],
  CNot[2]
 ];

Clear[ConstSum];
ConstSum[0]:=CNot[1];
ConstSum[1]:=GateArray[
  CNot[1],
  SelectedGate[QNot,{2}]
 ];

Clear[ConstAdder];
ConstAdder[1,b_Integer]:=Module[{a,c},
  a[i_]:=i;
  c[i_]:=i+2;
  GateArray[
   SelectedGate[ConstCarry[Mod[b,2]],{c[1],a[1],a[2]}],
   SelectedGate[If[b==1,QNot,IdentityGate],{a[1]}],
```

```
   SelectedGate[ConstSum[Mod[b,2]],{c[1],a[1]}]
  ]
 ];
ConstAdder[n_Integer,b_Integer]:=Module[{a,c},
  a[i_]:=i;
  c[i_]:=n+1+i;
  GateArray[
   SelectedGate[ConstCarry[Mod[b,2]],{c[1],a[1],c[2]}],
   SelectedGate[ConstAdder[n-1,Quotient[b,2]],
        Join[Table[a[i],{i,2,n+1}],Table[c[i],{i,2,n}]]],
   SelectedGate[InversedGate[ConstCarry[Mod[b,2]]],{c[1],a[1],c[2]}],
   SelectedGate[ConstSum[Mod[b,2]],{c[1],a[1]}]
  ]
 ];

Clear[ConstSubtractor];
ConstSubtractor[n_Integer,b_Integer]:=InversedGate[ConstAdder[n,b]];

Clear[QBitXor];
QBitXor[n_Integer]:=
  Apply[GateArray,Table[SelectedGate[CNot[1],{i,i+n}],{i,1,n}]];

Clear[ConstModuloAdder];
ConstModuloAdder[n_Integer,b_Integer,m_Integer]:=Module[{a1,a2,temp},
  a1=Range[1,n+1];
  temp=Range[Last[a1]+1,Last[a1]+n];
  a2=Range[Last[temp]+1,Last[temp]+n+1];
  GateArray[
   SelectedGate[ConstAdder[n,b],Join[a1,temp]],
   SelectedGate[QBitXor[n+1],Join[a1,a2]],
   SelectedGate[ConstSubtractor[n,m],Join[a1,temp]],
   SelectedGate[QNot,{Last[a1]}],
   SelectedGate[ControlledGate[ConstSubtractor[n,m],1],
        Join[{Last[a1]},a2,temp]],
   SelectedGate[QNot,{Last[a1]}],
   SelectedGate[ConstSubtractor[n,b],Join[a2,temp]],
   SelectedGate[QNot,{Last[a2]}],
   SelectedGate[ControlledGate[ConstAdder[n,m],1],Join[{Last[a2]},a1,temp]],
   SelectedGate[QNot,{Last[a2]}],
   SelectedGate[ConstAdder[n,b],Join[a2,temp]],
   SelectedGate[QBitXor[n+1],Join[a1,a2]]
  ]
 ];
```

```
Clear[ConstModuloSubtractor];
ConstModuloSubtractor[n_Integer,b_Integer,m_Integer]:=
  InversedGate[ConstModuloAdder[n,b,m]];

Clear[GetModuloInverse];
GetModuloInverse[1,m_Integer]:=1;
GetModuloInverse[a_Integer,m_Integer]:=
  Mod[(m*GetModuloInverse[Mod[m,a],a]-1)^2/a,m];

Clear[ConstModuloMultiplier];
ConstModuloMultiplier[n_Integer,c_Integer,m_Integer]:=Module[{a,c2,r,temp},
  a=Range[1,n];
  r=Range[Last[a]+1,Last[a]+n];
  temp=Range[Last[r]+1,Last[r]+2*n+2];
  c2=GetModuloInverse[c,m];
  GateArray[
   Apply[Sequence,Table[
     SelectedGate[ControlledGate[ConstModuloAdder[n,Mod[2^(i-1)*c,m],m],1],
      Join[{a[[i]]},r,temp]],{i,1,n}]],
   Apply[Sequence,Table[
     SelectedGate[
            ControlledGate[ConstModuloSubtractor[n,Mod[2^(i-1)*c2,m],m],1],
      Join[{r[[i]]},a,temp]],{i,1,n}]],
   SelectedGate[QBitXor[n],Join[r,a]],
   SelectedGate[QBitXor[n],Join[a,r]]
  ]
 ];

Clear[ConstModuloExp];
ConstModuloExp[n_Integer,x_Integer,m_Integer]:=Module[{a,r,temp},
  a=Range[1,n];
  r=Range[Last[a]+1,Last[a]+n];
  temp=Range[Last[r]+1,Last[r]+3*n+2];
  GateArray[
   SelectedGate[QNot,{r[[1]]}],
   Apply[Sequence,Table[
     SelectedGate[
            ControlledGate[ConstModuloMultiplier[n,PowerMod[x,2^(i-1),m],m],
              1],Join[{a[[i]]},r,temp]],
     {i,1,n}]]
  ]
 ];
```

```
Clear[ReversedDFTGate];
ReversedDFTGate[0]:=IdentityGate;
ReversedDFTGate[n_Integer/;n>0]:=Module[{R,S},
  R=MatrixGate[(ket[0].bra[0]+ket[0].bra[1]+ket[1].bra[0]-ket[1].bra[1])/
          Sqrt[2]];
  S[k_]:=SelectedGate[
    ControlledGate[
     MatrixGate[ket[0].bra[0]+ComplexExpand[Exp[I*Pi/2^(k-1)]]*ket[1].bra[1]],
          1
    ],{k,1}];
  GateArray[
   SelectedGate[ReversedDFTGate[n-1],Range[2,n]],
   Apply[Sequence,Table[S[i],{i,n,2,-1}]],
   R
  ]
 ];

Clear[BitReverseGate];
SetLinearity[BitReverseGate[n_Integer],BitReverseGate[n]];
BitReverseGate[n_Integer][r_ket]:=
  BitOrderArrangementGate[Join[Range[n,1,-1],Range[n+1,GetDimension[r]]]][r];

Clear[DFTGate];
DFTGate[n_Integer]:=GateArray[
  ReversedDFTGate[n],
  BitReverseGate[n]
 ];

Clear[FindOrderGate];
FindOrderGate[n_Integer,x_Integer,m_Integer]:=Module[{reg1,reg2,temp},
  reg1=Range[1,n];
  reg2=Range[Last[reg1]+1,Last[reg1]+n];
  temp=Range[Last[reg1]+1,Last[reg1]+3*n+2];
  GateArray[
   Apply[Sequence,Table[SelectedGate[RootNot,{reg1[[i]]}],{i,1,n}]],
   SelectedGate[ConstModuloExp[n,x,m],Join[reg1,reg2]],
   SelectedGate[MeasurementGate[n],reg2],
   SelectedGate[DFTGate[n],reg1],
   SelectedGate[MeasurementGate[n],reg1]
  ]
 ];
```

```
<<NumberTheory‘ContinuedFractions‘;
Clear[FindOrder];
FindOrder[n_Integer,x_Integer,m_Integer]:=Module[{reg,c,i,ord,temp},
  reg=Apply[ket,Table[0,{5*n+2}]];
  reg=FindOrderGate[n,x,m][reg];
  reg=Variables[reg][[1]];
  reg=SplitVector[reg,n][[1]];
  c=GetIndex[reg];
  For[i=1;ord=0,
      ord!=c/2^n&&(temp=Normal[ContinuedFraction[c/2^n,i]];
          Denominator[temp]<m),++i,ord=temp];
  Denominator[ord]
 ];
```

5.2.3 実行例

```
In[116]:=
Adder[8][ket[1,0,1,0,1,0,1,1]\[CircleTimes]ket[1,1,0,1,1,0,0,1,0]
        \[CircleTimes]ket[0,0,0,0,0,0,0,0]]//SplitVector[#,8,9]&
Out[116]=
{ket[1,0,1,0,1,0,1,1],ket[0,0,0,0,1,1,1,0,1],ket[0,0,0,0,0,0,0,0]}
In[120]:=
Subtractor[8][
    ket[1,0,1,0,1,1,0,0]\[CircleTimes]ket[1,0,0,1,1,0,0,0,0]\[CircleTimes]ket[
        0,0,0,0,0,0,0,0]]//SplitVector[#,8,9]&
Out[120]=
{ket[1,0,1,0,1,1,0,0],ket[0,0,1,0,0,1,1,1,1],ket[0,0,0,0,0,0,0,0]}
In[121]:=
ConstAdder[8,150][ket[1,0,0,1,0,1,0,0,0]\[CircleTimes]ket[0,0,0,0,0,0,0,0]]//
  SplitVector[#,9]&
Out[121]=
{ket[1,1,1,1,1,1,0,1,0],ket[0,0,0,0,0,0,0,0]}
In[122]:=
ConstSubtractor[8,50][
    ket[1,0,0,1,0,1,0,0,0]\[CircleTimes]ket[0,0,0,0,0,0,0,0]]//
  SplitVector[#,9]&
Out[122]=
{ket[1,1,1,0,1,1,1,1,1],ket[0,0,0,0,0,0,0,0]}
In[123]:=
ConstModuloAdder[8,10,11][
    ket[1,0,1,0,0,0,0,0,0]\[CircleTimes]ket[0,0,0,0,0,0,0,0,0]\[CircleTimes]ket[
        0,0,0,0,0,0,0,0,0]]//SplitVector[#,9]&
Out[123]=
```

```
{ket[0,0,1,0,0,0,0,0,0],ket[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]}
In[124]:=
ConstModuloSubtractor[8,10,11][
    ket[0,0,1,0,0,0,0,0,0]\[CircleTimes]ket[0,0,0,0,0,0,0,0]\[CircleTimes]ket[
        0,0,0,0,0,0,0,0,0]]//SplitVector[#,9]&
Out[124]=
{ket[1,0,1,0,0,0,0,0,0],ket[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]}
In[125]:=
ConstModuloMultiplier[4,3,7][
    ket[1,1,0,0]\[CircleTimes]ket[0,0,0,0]\[CircleTimes]ket[0,0,0,0,0,0,0,0,0,
        0]]//SplitVector[#,4]&
Out[125]=
{ket[0,1,0,0],ket[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]}
In[126]:=
ConstModuloExp[4,3,11][
    ket[1,1,0,0]\[CircleTimes]ket[0,0,0,0]\[CircleTimes]Apply[ket,
        Table[0,{3*4+2}]]]//SplitVector[#,4]&
Out[126]=
{ket[1,1,0,0],ket[1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]}
```

6 考察

量子コンピュータは古典コンピュータとは全く違う理論の上で動いている。即ち、古典コンピュータはブール代数を基礎としているのに対し、量子コンピュータがその礎にしているものは線形代数である。従って、古典コンピュータとは全く違った可能性を秘めている。しかし、実際に開発されているアルゴリズムはまだ多くなく、また、現実にはまだ数 qubit の量子コンピュータしか実現されていない。よって、量子コンピュータの実用化は、これらの点を如何にクリアするかということに掛かっているであろう。

参考文献

[1] Peter W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", quant-ph/9508027

[2] Adriano Barenco, et al, "Elementary gates for quantum computation", quant-ph/9503016

[3] "QUANTUM COMPUTER SIMULATOR", http://www.senko-corp.co.jp/qcs/

[4] 笠原皓司, 「線形代数学」, サイエンス社, 1982

[1],[2] は http://xxx.lanl.gov/archive/quant-ph からダウンロード可能である。