Microsoft Q# Coding Contest - Summer 2020 - Main Contest, June 19–22, 2020

Solutions to Problems E1 and E2

E1. POWER OF QUANTUM FOURIER TRANSFORM

Implement an operation that is equivalent to the operation QFT^P, where QFT is the quantum Fourier transform. Your operation should take the following inputs:

- an integer P, where $2 \le P \le 2.1 \cdot 10^6$,
- a register of type LittleEndian—a wrapper type for an array of qubits that encodes an unsigned integer in little-endian format, with the least significant bit written first (corresponding to array element with index 0).

The "output" of your solution is the state in which it left the input qubits.

Solution. Let $\mathrm{QFT}_{2^n} = \frac{1}{\sqrt{2^n}}[\omega_{2^n}^{k\ell}]_{k,\ell=0,\ldots,2^n-1}$, where $\omega_{2^n} = \exp(2\pi i/2^n)$ is a primitive 2^n complex root of unity. Note that QFT_{2^n} is the quantum Fourier transform as implemented in the Q# libraries, where the input and outputs are given in little-endian format.

We will first show that the operator QFT_{2^n} has a small multiplicative order, namely $\mathrm{QFT}_{2^n}^4 = \mathbf{1}_{2^n}$. To see this, denote $M = [m_{\ell,\ell'}] = \mathrm{QFT}_{2^n}$. We compute the coefficients of M as $m_{\ell,\ell'} = \sum_{k=0}^{2^n-1} \omega_{2^n}^{k(\ell+\ell')} = \delta_{\ell,2^n-\ell'}$. This means that M is a permutation matrix that looks like this:

$$M = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 1 \\ \vdots & & \iddots & \\ 0 & 1 & & \end{bmatrix}.$$

Note that M is an involution, i.e., $M^2 = \mathbf{1}_{2^n}$ which implies that $\mathrm{QFT}_{2^n}^4 = \mathbf{1}_4$. Therefore, for a given power $P \in \mathbb{N}$, we only have to compute the residue $p := P \bmod 4$ as $\mathrm{QFT}_{2^n}^P = \mathrm{QFT}_{2^n}^p$.

Listing 1. Power of the Quantum Fourier Transform

```
namespace Solution {
    open Microsoft.Quantum.Arithmetic;
    open Microsoft.Quantum.Intrinsic;
    open Microsoft.Quantum.Canon;
    open Microsoft.Quantum.Math;

    operation Solve (p : Int, inputRegister : LittleEndian) : Unit is Adj+Ctl {
        let r = ModulusI(p, 4);
        (OperationPowCA(QFTLE,r))(LittleEndian(inputRegister));
    }
}
```

E2. ROOT OF QUANTUM FOURIER TRANSFORM

Implement an operation that is equivalent to the operation $\mathrm{QFT}^{1/P}$, where QFT is the quantum Fourier transform. In other words, your operation, applied P times, should have the same effect as applying QFT. You can implement the required transformation up to a global phase. Your operation should take the following inputs:

- an integer P, where $2 \le P \le 8$,
- a register of type LittleEndian—a wrapper type for an array of qubits that encodes an unsigned integer in little-endian format, with the least significant bit written first (corresponding to array element with index 0).

The "output" of your solution is the state in which it left the input qubits.

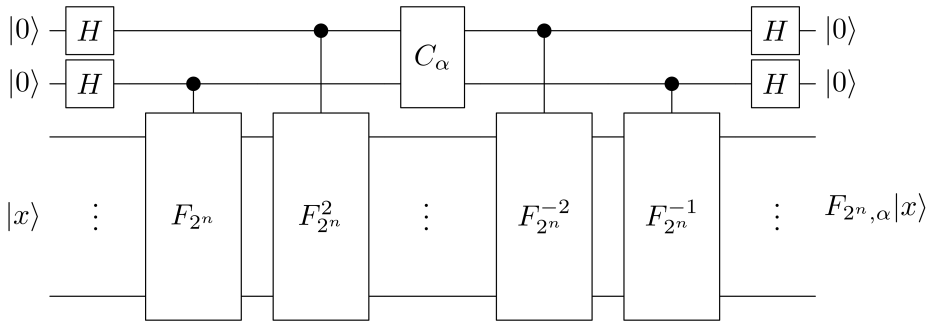

Figure 1. Quantum circuit realizing a fractional quantum Fourier transform

Solution. A matrix A having the property $A^\alpha = B$ with $\alpha \in \mathbb{R}$ is called an α-th root of B (where in general this root is not uniquely determined). In case of the discrete Fourier transform $B = \mathrm{QFT}$ we can define an α-th root of QFT_{2^n} via an ansatz

$$\mathrm{QFT}_{2^n,\alpha} := a_0(\alpha) \cdot \mathbf{1}_{2^n} + a_1(\alpha) F_{2^n} + a_2(\alpha) F_{2^n}^2 + a_3(\alpha) F_{2^n}^3, \tag{1}$$

where $F_{2^n} = \mathrm{QFT}_{2^n}$, as any function that is defined on the eigenvalue of an operator can be expressed as a Taylor series of the operator. As the order of F_{2^n} is four (as shown in Problem E1), the series can be re-arranged into the form given in eq. (**??**). Explicitly, we find that the coefficients $a_i(\alpha)$ for $i = 0, \ldots, 3$ are given by

$$\begin{aligned} a_0(\alpha) &:= \tfrac{1}{2}(\ \ 1+e^{i\alpha})\cos\alpha, \ \ a_1(\alpha) := \tfrac{1}{2}(\ \ 1-ie^{i\alpha})\sin\alpha, \\ a_2(\alpha) &:= \tfrac{1}{2}(-1+e^{i\alpha})\cos\alpha, \ \ a_3(\alpha) := \tfrac{1}{2}(-1-ie^{i\alpha})\sin\alpha. \end{aligned}$$

It is possible to show that the one-parameter family $\{\mathrm{QFT}_{2^n,\alpha} \mid \alpha \in \mathbb{R}\} \subset \mathbb{C}^{N\times N}$ has the following properties:

(i) $\mathrm{QFT}_{2^n,\alpha}$ is a unitary matrix for $\alpha \in \mathbb{R}$.

(ii) $\mathrm{QFT}_{2^n,0} = \mathbf{1}_{2^n}$ and $\mathrm{QFT}_{2^n,\pi/2} = \mathrm{QFT}_{2^n}$.

(iii) $(\mathrm{QFT}_{2^n,\alpha})^{\pi/(2\alpha)} = \mathrm{QFT}_{2^n}$ for $\alpha \in \mathbb{R}$.

We will use property (iii) for $\alpha = \pi/(2P)$ to obtain a unitary operator $R = \mathrm{QFT}_{2^n,\pi/(2P)}$ such that $R^P = \mathrm{QFT}_{2^n}$, i.e., R is a P-th root of the Fourier transform QFT_{2^n}.

The remaining question is how to find an implementation of $\mathrm{QFT}_{2^n,\alpha}$. To derive a circuit, we note that the additive decomposition in eq. (**??**) can be used to implement $\mathrm{QFT}_{2^n,\alpha}$ as shown in Fig. **??**. In this circuit there is a circulant matrix $C_\alpha := \mathrm{circ}(a_0(\alpha), \ldots, a_3(\alpha))$. Recall that a circulant matrix $\mathrm{circ}_G(a_0, \ldots, a_{n-1}) := (a_{j-i \bmod n})_{i,j=0,\ldots n-1}$ can be diagonalized by the Fourier transform. In our case, we find that

$$C_\alpha = \mathrm{circ}(a_0(\alpha), \ldots, a_3(\alpha)) = \mathrm{QFT}_4^{-1} \cdot \mathrm{diag}(1, e^{-i\alpha}, e^{2i\alpha}, e^{-i\alpha}) \cdot \mathrm{QFT}_4.$$

For more information about this decomposition, which is a special case of a linear combinations of unitaries (LCU) method for the case in which the operator has finite order, see [**?**].

Listing 2. Root of the Quantum Fourier Transform

```
namespace Solution {
    open Microsoft.Quantum.Arithmetic;
    open Microsoft.Quantum.Intrinsic;
    open Microsoft.Quantum.Canon;
    open Microsoft.Quantum.Math;
    open Microsoft.Quantum.Convert;

    operation Circ (qs : Qubit[], alpha : Double) : Unit is Adj + Ctl {
        let Q = FourierTransform_Reference;
        (Adjoint Q)(qs);
```

```
        ApplyDiagonalUnitary(
            [0.0, -alpha, -2.0*alpha, alpha],
            LittleEndian(qs)
        );
        Q(qs);
        }

    operation Solve (p : Int, inputRegister : LittleEndian) : Unit is Adj+Ctl {
        using (anc = Qubit[2]) {
            let Q = FourierTransform_Reference;
            let Q2 = OperationPowCA(Q, 2);
            within {
                ApplyToEachCA(H, anc);
                Controlled Adjoint Q ([anc[0]], inputRegister);
                Controlled Adjoint Q2([anc[1]], inputRegister);
            } apply {
                Circ(anc, PI() / (2.0 * IntAsDouble(k)));
            }
        }
    }
}
```

[1] A. Klappenecker and M. Roetteler. Quantum software reusability. *International Journal of Foundations of Computer Science*, 14(5):777–796, 2003. See also arXiv preprint quant-ph/0309121.