Microsoft Q# Coding Contest - Summer 2018 - Warmup Round
June 29 - July 2, 2018

Mariia Mykhailova and Chris Granade
Quantum Architectures and Computation Group, Microsoft Research, Redmond, WA, United States

TASKS AND SOLUTIONS

A. Generate plus state or minus state

You are given a qubit in state $|0\rangle$ and an integer *sign*.

Your task is to convert the given qubit to state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ if $sign = 1$ or $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ if $sign = -1$.

You have to implement an operation which takes a qubit and an integer as an input and has no output. The "output" of your solution is the state in which it left the input qubit.

Solution. Look up a list of quantum gates, for example, Quantum logic gate @ Wikipedia. One of the first gates there will be Hadamard - a gate which converts $|0\rangle$ into $|+\rangle$ and $|1\rangle$ into $|-\rangle$. Let's use it to create the required states.

The given qubit starts in $|0\rangle$ state, so for the case of $sign = 1$ it's sufficient to apply a Hadamard gate to it.

For the case of $sign = -1$, you need to convert the state of the given qubit to $|1\rangle$ before applying a Hadamard; this can be done by applying a Pauli X gate.

You can look up the syntax of conditional statements in Q# documentation.

Listing 1. Generate $|+\rangle$ or $|-\rangle$ state

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (q : Qubit, sign : Int) : ()
    {
        body
        {
            if (sign == -1) {
                X(q);
            }
            H(q);
        }
    }
}
```

B. Generate Bell state

You are given two qubits in state $|00\rangle$ and an integer *index*. Your task is to create one of the Bell states on them according to the *index*:

- $|B_0\rangle = \frac{1}{\sqrt{2}}\big(|00\rangle + |11\rangle\big)$
- $|B_1\rangle = \frac{1}{\sqrt{2}}\big(|00\rangle - |11\rangle\big)$
- $|B_2\rangle = \frac{1}{\sqrt{2}}\big(|01\rangle + |10\rangle\big)$
- $|B_3\rangle = \frac{1}{\sqrt{2}}\big(|01\rangle - |10\rangle\big)$

You have to implement an operation which takes an array of 2 qubits and an integer as an input and has no output. The "output" of your solution is the state in which it left the input qubits.

Solution. Let's start by generating $|B_0\rangle$ state. Since this state is an equal superposition of two basis states, we start by applying a Hadamard gate: $|00\rangle \to \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$. Now the first part of superposition matches the required state; to fix the second part, we need a two-qubit gate which change the state of the second qubit based on the state of the first one, i.e., will leave $|00\rangle$ unmodified and convert $|10\rangle$ to $|11\rangle$. This is CNOT gate.

Now that we have $|B_0\rangle$, we can modify it based on the value of *index* parameter to get the required state:

- If $index = 0$, we already have the required state, nothing else needs to be done.
- If $index = 1$, we need to flip the sign of $|11\rangle$ basis state; this can be done by applying Pauli Z gate to the first or the second qubit.
- If $index = 2$, we need to flip the state of the first or the second qubit; this can be done by applying Pauli X gate.
- Finally, if $index = 3$, we need to flip both the sign of one of the basis states and the state of one of the qubits.

We can unify this as follows: if $index \mod 2 = 1$, apply Z gate to flip the phase, and then if $index \div 2 = 1$, apply X gate to flip the state of one of the qubits. Both gates can be applied to either qubit.

Listing 2. Generate Bell state

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (qs : Qubit[], index : Int) : ()
    {
        body
        {
            H(qs[0]);
            CNOT(qs[0], qs[1]);
            // now we have (|00> + |11>) / sqrt(2) - modify it based on index arg
            if (index % 2 == 1) {
                // flip the phase of |11> component
                Z(qs[1]);
            }
            if (index / 2 == 1) {
                // flip the state of the second qubit
                X(qs[1]);
            }
        }
    }
}
```

Alternatively, you can follow the procedure described here: first convert $|00\rangle$ state to a different basis state using X gates on the proper qubits and then convert it to the proper Bell state using the sequence of Hadamard and CNOT gates. This approach will come in handy in problem E.

Listing 3. Generate Bell state

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (qs : Qubit[], index : Int) : ()
    {
        body
        {
            if (index % 2 == 1) {
                X(qs[0]);
            }
            if (index / 2 == 1) {
```

```
                X(qs[1]);
            }
            H(qs[0]);
            CNOT(qs[0], qs[1]);
        }
    }
}
```

C. Generate GHZ state

You are given N qubits ($1 \leq N \leq 8$) in zero state $|0...0\rangle$. Your task is to create Greenberger-Horne-Zeilinger (GHZ) state on them:

$$|GHZ\rangle = \frac{1}{\sqrt{2}}\big(|0...0\rangle + |1...1\rangle\big)$$

Note that for $N = 1$ and $N = 2$ GHZ state becomes states $|+\rangle$ and $|B_0\rangle$ from the previous tasks, respectively.

You have to implement an operation which takes an array of N qubits as an input and has no output. The "output" of your solution is the state in which it left the input qubits.

Solution. This problem is essentially just a generalized case of generating the first of Bell states. Start by applying Hadamard gate to one of the qubits (for simplicity the first one) to convert it to $|+\rangle$ state. If $N = 1$, you're done; otherwise you need to fix the states of the qubits starting with the second one to match the state of the first qubit; this is done by repeatedly applying CNOT gate with the first qubit at a control.

You can look up the syntax of for loop and array handling in Q# Programming Language guide.

Listing 4. Generate GHZ state

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (qs : Qubit[]) : ()
    {
        body
        {
            H(qs[0]);
            for (i in 1 .. Length(qs)-1) {
                CNOT(qs[0], qs[i]);
            }
        }
    }
}
```

D. Distinguish plus state and minus state

You are given a qubit which is guaranteed to be either in $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ or in $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ state.

Your task is to perform necessary operations and measurements to figure out which state it was and to return 1 if it was a $|+\rangle$ state or -1 if it was $|-\rangle$ state. The state of the qubit after the operations does not matter.

You have to implement an operation which takes a qubit as an input and returns an integer.

Solution. For an introduction to the concept of measurement, see The Qubit article. The qubit states that have to be distinguished are orthogonal (their scalar product $\langle +|-\rangle = 0$), and thus can be distinguished with certainity.

The computational basis measurement, implemented by Q# operation `M`, allows you to reliably distinguish states $|0\rangle$ and $|1\rangle$. Thus, you can use this measurement to distinguish $|+\rangle$ and $|-\rangle$ if you can come up with a transformation which maps $|+\rangle$ to $|0\rangle$ and $|-\rangle$ to $|1\rangle$ (or vice versa).

This transformation is the inverse (or, in quantum computing terms, adjoint) of the transformation discussed in problem A, which was done by a Hadamard gate. Since Hadamard gate is self-adjoint (it is its own adjoint), the necessary transformation is also done by a Hadamard gate.

After the original qubit state is transformed to $|0\rangle$ or $|1\rangle$, you can measure it using `M` operation; if the measurement result is `Zero`, the measured state is $|0\rangle$ and thus the original state was $|+\rangle$, otherwise the original state was $|-\rangle$.

Listing 5. Distinguish $|+\rangle$ and $|-\rangle$ states by transforming them into $|0\rangle$ and $|1\rangle$

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (q : Qubit) : Int
    {
        body
        {
            H(q);
            if (M(q) == Zero) {
                return 1;
            }
            return -1;
        }
    }
}
```

Alternatively, you can use a measurement which is not a computational basis measurement. For details on this, see Pauli measurements article. Note that $|+\rangle$ and $|-\rangle$ states are eigenstates of Pauli X matrix with eigenvalues +1 and -1, respectively. So measuring the given qubit in PauliX basis without any preliminary transformations will give the result: if the measurement result is `Zero`, the eigenvalue is +1 and thus the measured state is $|+\rangle$, otherwise the state is $|-\rangle$.

The operation which performs measurement in arbitrary Pauli basis instead of the default Pauli Z is `Measure`.

Listing 6. Distinguish $|+\rangle$ and $|-\rangle$ states using measurement in Pauli X basis

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (q : Qubit) : Int
    {
        body
        {
            if (Measure([PauliX], [q]) == Zero) {
                return 1;
            }
            return -1;
        }
    }
}
```

E. Distinguish Bell states

You are given two qubits which are guaranteed to be in one of the Bell states:

- $|B_0\rangle = \frac{1}{\sqrt{2}}\big(|00\rangle + |11\rangle\big)$
- $|B_1\rangle = \frac{1}{\sqrt{2}}\big(|00\rangle - |11\rangle\big)$
- $|B_2\rangle = \frac{1}{\sqrt{2}}\big(|01\rangle + |10\rangle\big)$

- $|B_3\rangle = \frac{1}{\sqrt{2}}\big(|01\rangle - |10\rangle\big)$

Your task is to perform necessary operations and measurements to figure out which state it was and to return the index of that state (0 for $|B_0\rangle$, 1 for $|B_1\rangle$ etc.). The state of the qubits after the operations does not matter.

You have to implement an operation which takes an array of two qubits as an input and returns an integer.

Solution. You need to find a way to convert $|B_0\rangle$ to $|00\rangle$, $|B_1\rangle$ to $|01\rangle$ etc. To do this, you can use the second solution to problem B, in which you converted basis states to Bell states. Since quantum gates are reversible and self-adjoint, you can apply them in reverse order to reverse the transformation. After that, you measure the qubits and convert the measurement results to an integer return.

Listing 7. Distinguish Bell states

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (qs : Qubit[]) : Int
    {
        body
        {
            CNOT(qs[0], qs[1]);
            H(qs[0]);
            return ResultAsInt([M(qs[0]); M(qs[1])]);
        }
    }
}
```

F. Distinguish multi-qubit basis states

You are given N qubits which are guaranteed to be in one of two basis states on N qubits. You are also given two bitstrings $bits0$ and $bits1$ which describe these basis states.

Your task is to perform necessary operations and measurements to figure out which state it was and to return 0 if it was the state described with $bits0$ or 1 if it was the state described with $bits1$. The state of the qubits after the operations does not matter.

You have to implement an operation which takes the following inputs:

- an array of qubits qs,
- two arrays of boolean values $bits0$ and $bits1$, representing the basis states in which the qubits can be. These arrays will have the same length as the array of qubits. $bits0$ and $bits1$ will differ in at least one position.

An array of boolean values represents a basis state as follows: the i-th element of the array is true if the i-th qubit is in state $|1\rangle$, and false if it is in state $|0\rangle$. For example, array [true; false] describes 2-qubit state $|10\rangle$.

Solution. This problem is actually a classical task in quantum disguise. Imagine that you are given two bitstrings $bits0$ and $bits1$, and a third one x which equals one of the two. How do you figure out which one it is, if you can only compare individual bits? You have to iterate over the bitstrings until you find the position in which their bits differ D (since the bitstrings are guaranteed to be different, you know such position exists). After that you compare the bits in that position $x[D]$ and $bits0[D]$; if they are equal, $x = bits0$, otherwise $x = bits1$.

Here you follow the same logic; the search of the position at which $bits0$ and $bits1$ is an entirely classical function, and you measure the qubit at that position using operation M to get Zero or One for false or true value of the bit, respectively.

Listing 8. Distinguish multi-qubit basis states

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;
```

```
    function FindFirstDiff (bits0 : Bool[], bits1 : Bool[]) : Int
    {
        for (i in 0 .. Length(bits1)-1) {
            if (bits0[i] != bits1[i]) {
                return i;
            }
        }
        return -1;
    }

    operation Solve (qs : Qubit[], bits0 : Bool[], bits1 : Bool[]) : Int
    {
        body
        {
            // find the first index at which the bitstrings are different and measure it
            let firstDiff = FindFirstDiff(bits0, bits1);
            let res = (M(qs[firstDiff]) == One);
            if (res == bits0[firstDiff]) {
                return 0;
            } else {
                return 1;
            }
        }
    }
}
```

G. Oracle for f(x) = k-th element of x

Implement a quantum oracle on N qubits which implements a function $f(x) = x_k$, i.e. the value of the function is the value of the k-th qubit.

You have to implement an operation which takes the following inputs:

- an array of qubits x (input register),
- a qubit y (output qubit),
- 0-based index of the qubit from input register k ($0 \leq k < \text{Length}(x)$).

The operation doesn't have an output; the "output" of your solution is the state in which it left the qubits.

An introduction to quantum oracles

An oracle O is a "black box" operation that is used as input to another algorithm. Often, such operations are defined using a classical function $f : \{0,1\}^n \to \{0,1\}^m$ which takes n-bit binary input and produces an m-bit binary output. To do so, consider a particular binary input $x = (x_0, x_1, \ldots, x_{n-1})$. We can label qubit states as $|\boldsymbol{x}\rangle = |x_0\rangle \otimes |x_1\rangle \otimes \cdots \otimes |x_{n-1}\rangle$.

We may first attempt to define O so that $O|x\rangle = |f(x)\rangle$, but this has a couple problems. First, f may have a different size of input and output ($n \neq m$), such that applying O would change the number of qubits in the register. Second, even if $n = m$, the function may not be invertible: if $f(x) = f(y)$ for some $x \neq y$, then $O|x\rangle = O|y\rangle$ but $O^\dagger O|x\rangle \neq O^\dagger O|y\rangle$. This means we won't be able to construct the adjoint operation $O^\dagger$, and oracles have to have an adjoint defined for them.

We can deal with both of these problems by introducing a second register of m qubits to hold our answer. Then we will define the effect of the oracle on all computational basis states: for all $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$,

$$O(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle\,.$$

Now $O = O^\dagger$ by construction, thus we have resolved both of the earlier problems.

Importantly, defining an oracle this way for each computational basis state $|x\rangle|y\rangle$ also defines how O acts for any other state. This follows immediately from the fact that O, like all quantum operations, is linear in the state that it acts on. Consider the Hadamard operation, for instance, which is defined by $H|0\rangle = |+\rangle$ and $H|1\rangle = |-\rangle$. If we wish to know how H acts on $|+\rangle$, we can use that H is linear,

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

In the case of defining our oracle O, we can similarly use that any state $|\psi\rangle$ on $n+m$ qubits can be written as

$$|\psi\rangle = \sum_{x\in\{0,1\}^n, y\in\{0,1\}^m} \alpha(x,y)|x\rangle|y\rangle ,$$

where $\alpha : \{0,1\}^n \times \{0,1\}^m \to \mathbb{C}$ represents the coefficients of the state $|\psi\rangle$. Thus,

$$\begin{aligned} O|\psi\rangle &= O \sum_{x\in\{0,1\}^n, y\in\{0,1\}^m} \alpha(x,y)|x\rangle|y\rangle \\ &= \sum_{x\in\{0,1\}^n, y\in\{0,1\}^m} \alpha(x,y) O|x\rangle|y\rangle \\ &= \sum_{x\in\{0,1\}^n, y\in\{0,1\}^m} \alpha(x,y)|x\rangle|y \oplus f(x)\rangle . \end{aligned}$$

Solution. In this case, we want to define an oracle for $f(x) = x_k$ as an operation O which transforms a state $|x\rangle|y\rangle$ into a state $|x\rangle|y \oplus x_k\rangle$ for any value of x that corresponds to a basis vector. Since this function has a single output bit ($m = 1$), our code has to flip the output register $|y\rangle$ when $x_k = 1$, and to do nothing if $x_k = 0$. That is,

$$\begin{aligned} O|x\rangle|y\rangle &= |x\rangle|y \oplus f(x)\rangle \\ &= |x\rangle|y \oplus x_k\rangle \\ &= |x\rangle \otimes \begin{cases} |y\rangle & \text{if } x_k = 0 \\ |\neg y\rangle & \text{if } x_k = 1 \end{cases} . \end{aligned}$$

Returning to the list of quantum gates, two-qubit operation `CNOT` is the gate which, when applied to k-th qubit of register x as control and qubit y as a target, performs exactly this transformation.

Listing 9. Oracle for $f(x) = x_k$

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (x : Qubit[], y : Qubit, k : Int) : ()
    {
        body
        {
            CNOT(x[k], y);
        }
    }
}
```

H. Oracle for f(x) = parity of the number of 1s in x

Implement a quantum oracle on N qubits which implements the following function: $f(x) = \sum_i x_i \mod 2$, i.e., the value of the function is 1 if x has odd number of 1s, and 0 otherwise.

You have to implement an operation which takes the following inputs:

- an array of qubits x (input register),
- a qubit y (output qubit).

The operation doesn't have an output; the "output" of your solution is the state in which it left the qubits.

Solution. The function $f(x) = \sum_i x_i \mod 2$ can be alternatively expressed as $f(x) = x_0 \oplus x_1 \oplus ... \oplus x_{n-1}$, i.e., a XOR of all elements of x. Since XOR of several values can be applied in any order, we can calculate $y \oplus f(x) = (...((y \oplus x_0) \oplus x_1)... \oplus x_{n-1})$.

As we've seen in the previous task, a CNOT gate applied to k-th element of register x as control and qubit y as target converts y to $y \oplus x_k$. So to calculate the required function we need to apply CNOT with each of the qubits of register x as controls.

Listing 10. Oracle for f(x) = parity of the number of 1s in x

```
namespace Solution {
    open Microsoft.Quantum.Primitive;
    open Microsoft.Quantum.Canon;

    operation Solve (x : Qubit[], y : Qubit) : ()
    {
        body
        {
            for (i in 0..Length(x)-1) {
                CNOT(x[i], y);
            }
        }
    }
}
```

I. Deutsch-Jozsa algorithm

You are given a quantum oracle - an operation on $N + 1$ qubits which implements a function $f : \{0, 1\}^N \to \{0, 1\}$. You are guaranteed that the function f implemented by the oracle is either constant (returns 0 on all inputs or 1 on all inputs) or balanced (returns 0 on exactly one half of the input domain and 1 on the other half).

There are only two possible constant functions: $f(x) = 0$ and $f(x) = 1$. The functions implemented by oracles in the two previous problems ($f(x) = x_k$ and $f(x) = \sum_i x_i \mod 2$) are examples of balanced functions.

Your task is to figure out whether the function given by the oracle is constant. Your code is allowed to call the given oracle only once.

You have to implement an operation which takes the following inputs:

- an integer N - the number of qubits in the oracle input,
- an oracle Uf, implemented as an operation with signature `((Qubit[], Qubit) => ())`, i.e., an operation which takes as input an array of qubits and an output qubit and has no output.

The return of your operation is a Boolean value: true if the oracle implements a constant function and false otherwise.

Solution. Deutsch-Jozsa algorithm is quite well described in the quantum computing literature, so we will not reproduce the description here. A good theoretical description can be found on Wikipedia.

For a Q# implementation, see the example at Microsoft/Quantum GitHub repository.