Quantum Computation & Cryptography
Lab sheet 2

In this lab you'll be working with 2 script files *Lab_2a.fsx*, *Lab_2b.fsx*. You will also be using the *blackBox.dll* file which is already imported by the scripts (check to see that the import path is correct). Check out the helper functions in each script file.

Part I

For this part you will only work in *Lab_2a.fsx*.

1. Implement the Deutsch-Josza algorithm as explained in the lectures. The template of your function should be: *deutschJosza (oracle:Qubits → unit) (qs:Qubits)*. It receives as input a list of qubits and an oracle function which acts on a list of qubits in the way described in the lectures. Your implementation should determine if this oracle function is constant or balanced. You can test your implementation with 5 black-box functions provided in the *blackBox.dll* library (which is already imported in the script file). The black-box functions are *BlackBox.fun1*, *BlackBox.fun2*, *BlackBox.fun3*, *BlackBox.fun4*, *BlackBox.fun5*. Note that each function works with only a specific number of qubits (since if these were generic constant/balanced functions you could just test them on one qubit :)). The number of qubits for each function, in order, are: 5, 7, 10, 6, 8. Here's an example call:

```
let k = Ket(5)
let qs = k.Qubits
deutschJosza BlackBox.fun1 qs
```

2. Implement a version of the Elitzur-Vaidman bomb detector. You are given a 2-qubit quantum gate as a black-box. You are promised that this gate is either CNOT (bomb) or identity (dud) and you have to determine which one. You can only use 2 qubits, which initially start in the $|00\rangle$ state and are allowed only local operations on these qubits (i.e. no 2-qubit gates other than the one provided) and computational basis measurements. You can repeat any of these operations as many times as you like. But there's a catch: if you measure the second qubit as being $|1\rangle$, you explode. Your solution must have arbitrarily small probability of triggering the explosion and, conditioned on not exploding, to perfectly distinguish between bomb and dud. You can test your implementation with the candidate functions: *BlackBox.candidate1* - *BlackBox.candidate6*. Here's an example call:

```
let k = Ket(2)
```

```
let qs = k.Qubits
bombDetector BlackBox.candidate3 qs
```

This problem and the solution are based on [1].

Part II

For this part you will only work in *Lab_2b.fsx*.

1. Implement Grover's algorithm as explained in the slides. The template of your function should be: *grover (oracle:Qubits → unit) (qs:Qubits)*. It receives as input a list of qubits and an oracle function which acts on a list of qubits in the way described in the slides (note that unlike the slides, the oracle does not act on an additional qubit). Your implementation should find the hidden value of the oracle. You can test your implementation with 5 black-box functions provided in the *blackBox.dll* library (which is already imported in the script file). The black-box functions are *BlackBox.oracle1* - *BlackBox.oracle5*. Again these functions require a minimal number of qubits to represent the solution. For all these functions the minimal number is 4. Since the algorithm is probabilistic, you should perform multiple runs to determine the solution (the majority output). Here's an example call:

```
let k = Ket(4)
for i in 1 .. 20 do
   let qs = k.Reset(4)
   grover BlackBox.oracle4 qs
   printFromBinary qs
```

Additionally, there is one more oracle function called *BlackBox.bigOracle*. This function requires 11 qubits and so your implementation will probably be very slow. To get around this, turn your function into a circuit (as in the previous lab) and use the optimization function *GrowGates* as follows:

```
let k = Ket(11)
let qs = k.Qubits
let qcirc = Circuit.Compile (grover BlackBox.bigOracle) qs
let qcirc = qcirc.GrowGates(k)
for i in 1 .. 20 do
   let qs = k.Reset(11)
   qcirc.Run qs
   printFromBinary qs
```

Compare the performances of the two approaches (with and without optimization).

2. Implement the *quantum SWAP test*. Given two unknown qubits $|\psi\rangle$ and $|\phi\rangle$ we want to determine whether they are close to each other (almost the same state) or far from each other (almost orthogonal). To do this we use the following procedure: use a third qubit initially in the $|+\rangle$ state. Perform a

controlled-SWAP operation[1] using this qubit as control and $|\psi\rangle$ and $|\phi\rangle$ as the target qubits. Perform a H on the control qubit and measure it in the computational basis. If the control qubit is 0 accept (i.e. say that the states are equal), otherwise reject. What is the acceptance probability? Implement this functionality and test it.

References

[1] Aharon Brodutch, Daniel Nagaj, Or Sattath, Dominique Unruh *An adaptive attack on Wiesner's quantum money.* `https://arxiv.org/abs/1404.1507`.

[1] This operation swaps the two target qubits if the control is $|1\rangle$, otherwise does nothing. Liquid already has the SWAP operation. To make it controlled, check the documentation.