G53NSC and G54NSC
Non-Standard Computation
Lab 4 Exercises

Dr. Alexander S. Green

18th February 2010

Exercise sheet 4

These exercises carry on from Exercise sheets 1, 2 and 3; and may use some of the types and functions you have previously defined.

The exercises are listed here, but more information can be found in the hints and tips section below.

1. Implement, in Haskell and *QIO*, the EPR experiment, as was shown in the lecture slides.
2. Implement a function ($experiments :: Int \to IO\ (Float, Float)$) that repeatedly *runs* the EPR experiment, the number of times given by the argument, and returns the fraction of times that the measurements agree for both the classical and quantum variant of the experiment
3. What fraction of the experiments give matching measurements if you run the experiment 100,000 times? What does this suggest to us about quantum computation?
4. Implement a function $ghz :: Int \to QIO\ [Qbit]$ that creates a GHZ state over the number of qubits given by the first argument.
5. Implement a function, $oneTenth :: QIO\ Bool$, that returns a *True* value only with probability $\frac{1}{10}$.
6. Implement a function, $falseProb :: Float \to QIO\ Bool$, that returns a *False* value only with the probability of the argument.
7. In the lecture, we have only been looking at one of the Bell states. There are actually four of them. Implement in *QIO*, functions that return a pair of qubits in each of the other three Bell states.
8. Implement a function $bellMeasurement :: (Qbit, Qbit) \to QIO\ (Bool, Bool)$ that does a Bell measurement of the two argument qubits.

Hints and Tips

Exercise information

1. You can use all the definitions given in the slides, but may need to define extra functions as well, e.g. the *randomSetting* :: *IO Setting* function needs to be defined.

2. The *experiment* function defined for the previous question returns a pair of Booleans which are whether or not the measurements agreed in the classical and quantum variant of the experiment. The function *experiments* that you need to define, must keep track of what percentage of the runs returned *True* for either variant. Try to make your solution efficient so the next question doesn't take too long.

3. This should just be the result returned when you run *experiments* 100000 (a good implementation should be able to do this in less than a minute, and without a stack overflow!), and a brief comment on what these values seem to imply.

4. A GHZ state is a generalisation of the Bell state that we have been using. It generalises the sharing of the $|+\rangle$ state over an arbitrary number of qubits (> 2). E.g. the GHZ state over three qubits is $\frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle$. You may find this exercise easier if you implement a function *share* :: *Qbit* $\rightarrow$ *QIO Qbit* that *shares* the state of the argument qubit with a newly initialised qubit, and returns the new qubit.

5. In order to do this, you need to rotate a single qubit the correct amount so that its measurement probabilities are as required, e.g. *True* or $|1\rangle$ with probability $\frac{1}{10}$. The best way to do this is to define an exponentiation of one of the Pauli gates. E.g. A Pauli gate can be exponentiated so that it represents a rotation by any angle θ around its corresponding axis. For Pauli-X and Pauli-Y gates, the exponentiations can be defined as follows for any angle θ.

$$R_x(\theta) = \begin{bmatrix} cos\frac{\theta}{2} & -isin\frac{\theta}{2} \\ -isin\frac{\theta}{2} & cos\frac{\theta}{2} \end{bmatrix} \qquad R_y(\theta) = \begin{bmatrix} cos\frac{\theta}{2} & -sin\frac{\theta}{2} \\ sin\frac{\theta}{2} & cos\frac{\theta}{2} \end{bmatrix}$$

   The trick is to then find an angle that takes one of the base states to a state which has the correct measurement properties. You should be able to calculate the angle.

6. This exercise is similar to the previous exercise, but the rotation required must now be calculated from the input. You can assume that the input argument n is always in the range $0 \leqslant n \leqslant 1$. Because *QIO* uses floating point numbers to simulate the complex amplitudes, you will find that your implementation is not exact. E.g. It doesn't matter if simulating your function gives results that aren't exactly the input probability.

As a guide, the following is what my implementation gives:

$> \ \mathit{sim}\ (\mathit{falseProb}\ 0.2)$
$[(\mathit{True}, 0.79999995), (\mathit{False}, 0.20000005)]$

7. The four Bell states are as follows:

$$\frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}\,|11\rangle \qquad \frac{1}{\sqrt{2}}\,|00\rangle - \frac{1}{\sqrt{2}}\,|11\rangle$$

$$\frac{1}{\sqrt{2}}\,|01\rangle + \frac{1}{\sqrt{2}}\,|10\rangle \qquad \frac{1}{\sqrt{2}}\,|01\rangle - \frac{1}{\sqrt{2}}\,|10\rangle$$

8. The Bell measurement is an important concept that we shall be using when we look at quantum teleportation, and super-dense coding. It can be thought of as measuring which of the four Bell states a pair of qubits are in (if they're in a Bell state). That is it maps the four Bell states to the four two-qubit base states. You should define a member of the U data-type that does this transformation (e.g. $\mathit{bellU} :: \mathit{Qbit} \rightarrow \mathit{Qbit} \rightarrow U$), and then measure the qubits. If the inputs were in a Bell state, then the output will be one of the two-qubit base states with probability 1. Try it out with the four Bell states that you defined for the previous question.

The Quantum IO Monad

The Quantum IO Monad, or *QIO* is a monadic interface from Haskell to quantum computation. More precisely, it is a library that allows you to define unitary operators and effectful quantum computations, along with simulator functions that allow you to *run* the quantum computations that you define. A lot of information on *QIO* including its implementation are available online (see the links on the course webpage). Installation of *QIO* is relatively straightforward if you can make use of cabal (cabal is part of the Haskell platform, and as such should already be installed on the machines in A32).

The following list of instructions will install *QIO* on the windows machines in A32 (but you may need to re-install it for every session). The following commands should be entered in a command prompt:

- Set the http proxy in the current command prompt

```
set HTTP_PROXY=wwwcache.cs.nott.ac.uk:3128
```

- Make sure the cabal list of packages is up to date:

```
cabal update
```

- Install *QIO* (in your own user space, as you don't have global permissions)

```
cabal install QIO --user
```

(note: if you don't have a proxy, and you are using your own machine, then you should just have to update the list of packages as above, and install the *QIO* package without the –user flag)

If you are having difficulties installing *QIO* you can always download the source from: http://www.cs.nott.ac.uk/ asg/QIO/ and import the files as necessary. However, i would recommend this as a last resort, and suggest that you contact me for support.

Information

The exercises set in the labs have a firm deadline of 12:00 (midday); Thursday the 1st of April, but it is highly recommended that you submit your work on a weekly basis (E.g. 1 week after the date each exercise sheet is released) to enable you to receive ongoing feedback. I will give feedback for any exercises submitted within 2 weeks of their original release date.

The weekly submissions should be emailed to me (asg@cs.nott.ac.uk), or handed to me in the labs. The final submission of your portfolio will be through the school office by 12:00 (midday) on Thursday the 1st of April (The last day of the Spring term). The final submission through the school office should be made even if you have been submitting work to me on a weekly basis as it is this final submission that counts as your portfolio.

These exercise sheets should be attempted on your own, and at the end of the course, it is these individual submissions that will make up your portfolio project. Combined, the work submitted in your portfolio is worth 50% of the mark for this module (The other 50% consisting of the research report and presentation).