G53NSC and G54NSC
Non-Standard Computation
Lab 5 Exercises

Dr. Alexander S. Green

25th February 2010

Exercise sheet 5

These exercises carry on from Exercise sheets 1, 2, 3 and 4; 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 superdense coding in *QIO*.
2. Implement quantum teleportation in *QIO*.
3. Implement Deutsch's algorithm in *QIO*.
4. How many functions of type $(Bool, Bool, Bool) \rightarrow Bool$ are constant, and how many are balanced? Define some of them in Haskell.
5. Implement a variant of the Deutsch-Jozsa algorithm that works for the functions defined for the previous exercise, E.g. that takes a function in $(Bool, Bool, Bool) \rightarrow Bool$ as its input.

Hints and Tips

Exercise information

1. Superdense coding can be thought of in three parts. First, there is the creation of a Bell pair, then there is what Alice does with her classical data, and one member of the Bell pair, and then finallly what Bob does to extract the original classical data. You should already have a function that implements a Bell pair in *QIO*, so just need to define a function $sdcAlice :: (Bool, Bool) \rightarrow Qbit \rightarrow QIO\ Qbit$ that represents Alice applying the relevent unitary operation to the given qubit, and a function $sdcBob :: Qbit \rightarrow Qbit \rightarrow QIO\ (Bool, Bool)$ which represents what Bob has to do.

Finally, you will have to piece them together to give a function *superdense*:: $(Bool, Bool) \rightarrow QIO\ (Bool, Bool)$

2. Quantum teleportation can also be thought of in three parts. Again, the first part sets up a Bell pair, the second part is what Alice has to do, and the final part is what Bob has to do. You should define a function *qtAlice*:: $Qbit \rightarrow Qbit \rightarrow QIO\ (Bool, Bool)$, and a function $qtBob :: (Bool, Bool) \rightarrow Qbit \rightarrow QIO\ Qbit$, and then put them together with the creation of a Bell state in an overall function $teleport :: Qbit \rightarrow QIO\ Qbit$

   Note. You should be able to define the functions *qtAlice* and *qtBob* in terms of the functions *sdAlice*, and *sdBob*.

   Try and think of a good way of testing your *teleport* function, maybe defining something like the *falseProb* exercise from last week, that teleports the qubit before measurement.

3. The overall function you define for this exercise should have the type $deutsch :: (Bool \rightarrow Bool) \rightarrow QIO\ Bool$, that is, it takes a Boolean function as its argument, and returns a Boolean value that represents whether or not the given function is balanced.

   The unitary required for Deutsch's algorithm can be defined using the input Boolean function.

4. There are lots more balanced functions of type $(Bool, Bool, Bool) \rightarrow Bool$ than constant ones. I would suggest defining 2 constant functions, and 6 balanced functions.

5. The Deutsch-Jozsa is very similar to Deutsch's algorithm, again, you should construct the required unitary from the input function. The overall function you define, should have the type *deutschJozsa*::$((Bool, Bool, Bool) \rightarrow Bool) \rightarrow QIO\ (Bool, Bool, Bool)$. If all three output Booleans are False, then the input function was constant.

   You may find the following function useful:

$$
\begin{aligned}
&cond3 :: (Qbit, Qbit, Qbit) \rightarrow ((Bool, Bool, Bool) \rightarrow U) \rightarrow U \\
&cond3\ (q0, q1, q2)\ f = cond\ q0\ (\lambda x \rightarrow \\
&\qquad\qquad cond\ q1\ (\lambda y \rightarrow \\
&\qquad\qquad\quad cond\ q2\ (\lambda z \rightarrow \\
&\qquad\qquad\qquad f\ (x, y, z))))
\end{aligned}
$$

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).