Problem Set 3 - Simulation of quantum circuits

Computação Quântica 2018/2019

March 18, 2019

This set requires the module `System.Random`, from Haskell's libraries, to be imported. Documentation is available here. Modules `Data.Complex` and `Data.List` may also be helpful in solving the problem set.

Consider the column vector representation of a multi-qubit quantum state. Each entry of the vector corresponds to the complex coefficient of an associated orthonormal basis state. For the quantum circuit model of computation, the computational basis is typically used; it is obtained by picking the same basis for each qubit, taking the basis which is the tensor product of all the (individual qubit) bases. As an example, admit a general two-qubit state $|q_0q_1\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \zeta|11\rangle$. After measurement, the state will collapse to any of the basis states with nonzero amplitude.

$$|q_0q_1\rangle \rightarrow \begin{bmatrix}\alpha\\ \beta\\ \gamma\\ \zeta\end{bmatrix} \begin{cases}|00\rangle\\ |01\rangle\\ |10\rangle\\ |11\rangle\end{cases} \overset{measurement}{\rightarrow} \begin{bmatrix}0\\ 1\\ 0\\ 0\end{bmatrix} \begin{cases} \\ |\mathbf{01}\rangle\\ \\ \end{cases}$$

In the above example, the state collapsed into $|01\rangle$ - this event had a β^2 probability of occurrence. The column vector describing the measured state reflects that change (the kets to the right of each vector are presented for better comprehension).

1. Keeping the above information in mind, and knowing that the `System.Random` module of Haskell allows for the generation of a pseudorandom `Float` between 0 and 1, we can start building a state measurement simulator.

   (a) Build a function `amplitude_acc :: [[Complex Float]] -> [Float]` to create a list of `Float` corresponding to the cumulative squared values of the coefficients of an input quantum state. Example:

   `Input: [[`α`],[`β`]] ; Output: [`α^2` , `$\alpha^2 + \beta^2$`]`

(b) Build a function `meas_acc :: [Float] -> Float -> [Float]`, that takes a list output by `amplitude_acc` and a `Float` (which should be random, but we'll handle that later), and returns a string corresponding to the basis state associated with the interval in which the `Float` falls relative to the accumulated list. Example:

```
Inputs:  [0.5 , 1.0] 0.7 ;  Output:  [0.0 , 1.0]
```

Note: it may be easier to visualize the input list as containing the upper limit of the interval it contains, with the lower limit defined by the previous entry of the list. The above list contains the interval [0, 0.5[in its first entry, and [0.5 , 1.0[in the second one.

(c) Implement a function `state_to_char :: [Float] -> [Char]` that takes a measured quantum state (i.e. the output of `meas_acc`) and returns a string describing the corresponding state ket. Example:

```
Input:  [0.0 , 1.0, 0.0, 0.0] ;  Output:  "01"
```

(d) Verify that the previous functions work by implementing an IO function `meas`:

```
meas :: [[Complex Float]] -> IO [Char]
meas x = do
  n <- randomIO :: IO Float
  return $ state_to_char $ meas_acc (amplitude_acc x) n
```

This function takes any quantum state in the form `[[Complex Float]]` and outputs a string describing the measured basis state. Apply the function multiple times to the same superposition state (obtained for example by applying the Hadamard gate to $|0\rangle$), and check the results.

2. A single measurement of a quantum state does not give much information other than the resulting state having a non-zero probability of occurring. The study of quantum circuit measurements, either in a simulator or a quantum device, typically requires a great number of executions (also known as shots) and associated results, to accurately determine probability amplitudes of basis states.

(a) Implement a function `shots :: [[Complex Float]] -> Int -> IO [[Char]]` that takes a quantum state and an `Int` n, and returns a list containing n measurement strings. Example (`had`, `‘p‘` and `q0` were defined in previous classes):

```
Inputs:  (had ‘p‘ q0) 4 ;  Output:  ["0", "1", "0", "0"]
```

(b) For better visualization of simulation results, implement a function

```
freqs ::  [[Complex Float]] -> Int -> IO [([Char], Int)]
```

that, like `shots`, takes a quantum state and an `Int` n. This function outputs measurement results as a tuple containing the measured state, and the number of times it occurred. Example:

```
Inputs:  (had ‘p‘ q0) 100 ;  Output:  [("0",54),("1",46)]
```

Note: alternatively, implement a function `freq ::IO [[Char]] -> IO [([Char], Int)]` that takes a list of measured results (i.e. the output of `shots`) and converts it into a tuple containing the measured state and the number of times it occurred.