Algebra of Quantum Operations

Problem Set 3

Ana Neri

March 12, 2021

The goal of the problem set 3 is to give intuition about quantum circuit simulations in Haskell.

You can find the documentation page for the Data.Random Haskell package helpful to generate pseudo random Float.

Let's recall some essential concepts

Qubit System

Consider the column vector representation of a multi-qubit quantum state. The characterisation of a quantum state with n qubits is a vector with a dimension of 2^n. Each entry of the vector corresponds to the complex coefficient of an associated orthonormal basis state. As an example, admit a general two-qubit state $|q_0 q_1\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \sigma|11\rangle$. After measurement, the state will collapse to any of the basis states with nonzero amplitude.

$$|q_o q_1\rangle = \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \zeta \end{pmatrix} \begin{cases} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{cases} \xrightarrow{measurement} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \Big\{ |01\rangle$$

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. (a) Build a function `amplitude_acc` 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`, 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 visualise the input list as the upper limit of its interval, 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` 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` that takes a quantum state and an Int n, and returns a list containing n measurement strings. Example (had, 'tensor' and q0 were defined in previous classes):
   Inputs: (had 'tensor' q0) 4 ; Output: ["0", "1", "0", "0"]

   (b) For better visualisation of simulation results, implement a function `freqs` 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 'tensor' q0) 100 ; Output: [("0",54),("1",46)]`