**1. Initialize the environment and set global variables**

Here I initialize the environment with global variables and then display a few of them.

rm(list=ls())

#Call the init function to initialize the environment and create global variables

init()

# Display some of global variables in environment

ls()

## [1] "I16" "I2" "I4" "I8" "q0\_" "q00\_" "q000\_"

## [8] "q0000\_" "q00000\_" "q00001\_" "q0001\_" "q00010\_" "q00011\_" "q001\_"

## [15] "q0010\_" "q00100\_" "q00101\_" "q0011\_" "q00110\_" "q00111\_" "q01\_"

## [22] "q010\_" "q0100\_" "q01000\_" "q01001\_" "q0101\_" "q01010\_" "q01011\_"

## [29] "q011\_" "q0110\_" "q01100\_" "q01101\_" "q0111\_" "q01111\_" "q1\_"

## [36] "q10\_" "q100\_" "q1000\_" "q10000\_" "q10001\_" "q1001\_" "q10010\_"

## [43] "q10011\_" "q101\_" "q1010\_" "q10100\_" "q10101\_" "q1011\_" "q10110\_"

## [50] "q10111\_" "q11\_" "q110\_" "q1100\_" "q11000\_" "q11001\_" "q1101\_"

## [57] "q11010\_" "q11011\_" "q111\_" "q1110\_" "q11100\_" "q11101\_" "q1111\_"

## [64] "q11110\_" "q11111\_"

#1. 2 x 2 Identity matrix

I2

## [,1] [,2]

## [1,] 1 0

## [2,] 0 1

#2. 8 x 8 Identity matrix

I8

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]

## [1,] 1 0 0 0 0 0 0 0

## [2,] 0 1 0 0 0 0 0 0

## [3,] 0 0 1 0 0 0 0 0

## [4,] 0 0 0 1 0 0 0 0

## [5,] 0 0 0 0 1 0 0 0

## [6,] 0 0 0 0 0 1 0 0

## [7,] 0 0 0 0 0 0 1 0

## [8,] 0 0 0 0 0 0 0 1

#3. Qubit |00>

q00\_

## [,1]

## [1,] 1

## [2,] 0

## [3,] 0

## [4,] 0

#4. Qubit |010>

q010\_

## [,1]

## [1,] 0

## [2,] 0

## [3,] 1

## [4,] 0

## [5,] 0

## [6,] 0

## [7,] 0

## [8,] 0

#5. Qubit |0100>

q0100\_

## [,1]

## [1,] 0

## [2,] 0

## [3,] 0

## [4,] 0

## [5,] 1

## [6,] 0

## [7,] 0

## [8,] 0

## [9,] 0

## [10,] 0

## [11,] 0

## [12,] 0

## [13,] 0

## [14,] 0

## [15,] 0

## [16,] 0

#6. Qubit 10010

q10010\_

## [,1]

## [1,] 0

## [2,] 0

## [3,] 0

## [4,] 0

## [5,] 0

## [6,] 0

## [7,] 0

## [8,] 0

## [9,] 0

## [10,] 0

## [11,] 0

## [12,] 0

## [13,] 0

## [14,] 0

## [15,] 0

## [16,] 0

## [17,] 0

## [18,] 0

## [19,] 1

## [20,] 0

## [21,] 0

## [22,] 0

## [23,] 0

## [24,] 0

## [25,] 0

## [26,] 0

## [27,] 0

## [28,] 0

## [29,] 0

## [30,] 0

## [31,] 0

## [32,] 0

The QCSimulator implements the following gates

1. Pauli X,Y,Z, S,S’, T, T’ gates
2. Rotation , Hadamard,CSWAP,Toffoli gates
3. 2,3,4,5 qubit CNOT gates e.g CNOT2\_01,CNOT3\_20,CNOT4\_13 etc
4. Toffoli State,SWAPQ0Q1

**2. To display the unitary matrix of gates**

To check the unitary matrix of gates, we need to pass the appropriate identity matrix as an argument. Hence below the qubit gates require a 2 x 2 unitary matrix and the 2 & 3 qubit CNOT gates require a 4 x 4 and 8 x 8 identity matrix respectively

PauliX(I2)

## [,1] [,2]

## [1,] 0 1

## [2,] 1 0

Hadamard(I2)

## [,1] [,2]

## [1,] 0.7071068 0.7071068

## [2,] 0.7071068 -0.7071068

S1Gate(I2)

## [,1] [,2]

## [1,] 1+0i 0+0i

## [2,] 0+0i 0-1i

CNOT2\_10(I4)

## [,1] [,2] [,3] [,4]

## [1,] 1 0 0 0

## [2,] 0 0 0 1

## [3,] 0 0 1 0

## [4,] 0 1 0 0

CNOT3\_20(I8)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]

## [1,] 1 0 0 0 0 0 0 0

## [2,] 0 0 0 0 0 1 0 0

## [3,] 0 0 1 0 0 0 0 0

## [4,] 0 0 0 0 0 0 0 1

## [5,] 0 0 0 0 1 0 0 0

## [6,] 0 1 0 0 0 0 0 0

## [7,] 0 0 0 0 0 0 1 0

## [8,] 0 0 0 1 0 0 0 0

**3. Compute the inner product of vectors**

For example of phi = 1/2|0> + sqrt(3)/2|1> and si= 1/sqrt(2)(10> + |1>) then the inner product is the dot product of the vectors

phi = matrix(c(1/2,sqrt(3)/2),nrow=2,ncol=1)

si = matrix(c(1/sqrt(2),1/sqrt(2)),nrow=2,ncol=1)

angle= innerProduct(phi,si)

cat("Angle between vectors is:",angle)

## Angle between vectors is: 15

**4. Compute the dagger function for a gate**

The gate dagger computes and displays the transpose of the complex conjugate of the matrix

TGate(I2)

## [,1] [,2]

## [1,] 1+0i 0.0000000+0.0000000i

## [2,] 0+0i 0.7071068+0.7071068i

GateDagger(TGate(I2))

## [,1] [,2]

## [1,] 1+0i 0.0000000+0.0000000i

## [2,] 0+0i 0.7071068-0.7071068i

**5. Invoking gates in series**

The Quantum gates can be chained by passing each preceding Quantum gate as the argument. The final gate in the chain will have the qubit or the identity matrix passed to it.

# Call in reverse order

# Superposition states

# |+> state

Hadamard(q0\_)

## [,1]

## [1,] 0.7071068

## [2,] 0.7071068

# |-> ==> H x Z

PauliZ(Hadamard(q0\_))

## [,1]

## [1,] 0.7071068

## [2,] -0.7071068

# (+i) Y ==> H x S

SGate(Hadamard(q0\_))

## [,1]

## [1,] 0.7071068+0.0000000i

## [2,] 0.0000000+0.7071068i

# (-i)Y ==> H x S1

S1Gate(Hadamard(q0\_))

## [,1]

## [1,] 0.7071068+0.0000000i

## [2,] 0.0000000-0.7071068i

# Q1 -- TGate- Hadamard

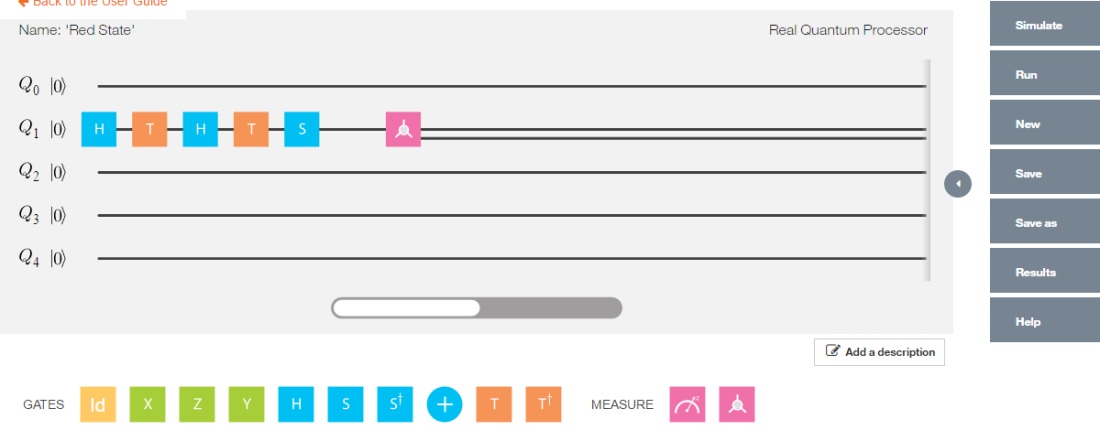
Q1 = Hadamard(TGate(I2))

**6. More gates in series**

**TGate of depth 2**

The Quantum circuit for a TGate of Depth 2 is

Q0 — Hadamard-TGate-Hadamard-TGate-SGate-Measurement as shown in IBM’s Quantum Experience Composer

[](https://gigadom.wordpress.com/2016/06/23/introducing-qcsimulator-a-5-qubit-quantum-computing-simulator-in-r/untitled-36/)

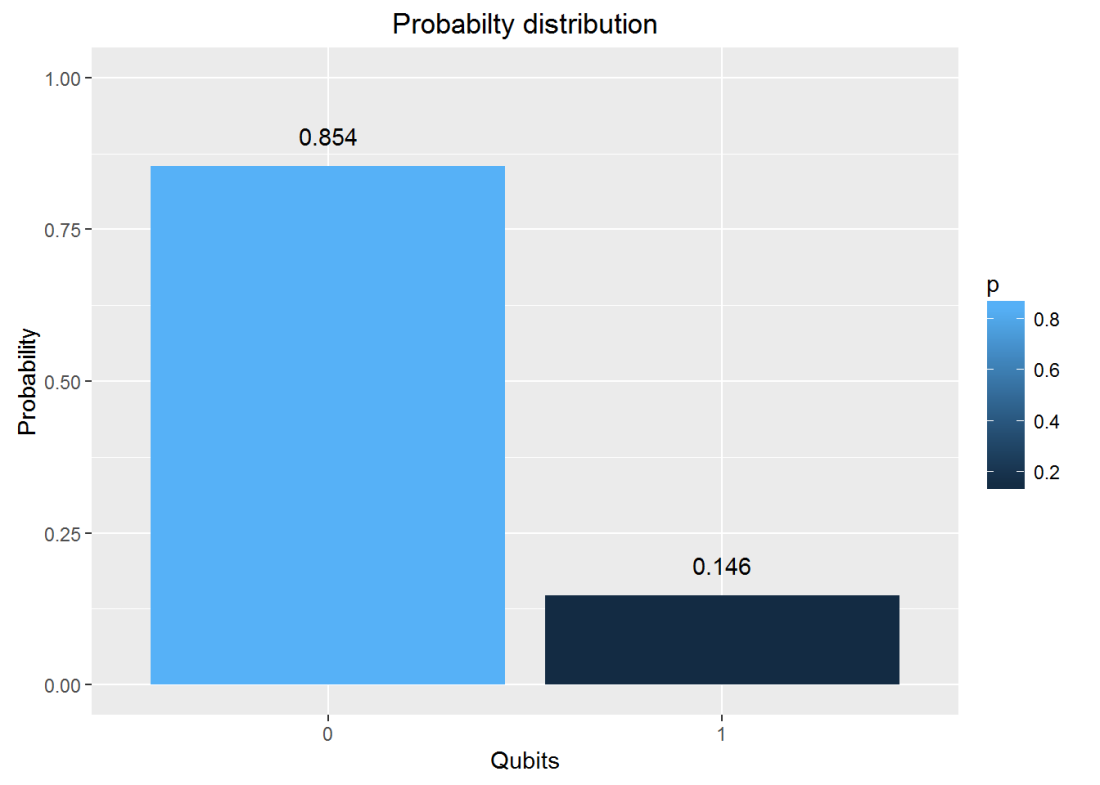
Implementing the quantum gates in series in reverse order we have

# Invoking this in reverse order we get

a = SGate(TGate(Hadamard(TGate(Hadamard(q0\_)))))

result=measurement(a)

plotMeasurement(result)

[](https://gigadom.wordpress.com/2016/06/23/introducing-qcsimulator-a-5-qubit-quantum-computing-simulator-in-r/fig0-1/)

**7. Invoking gates in parallel**

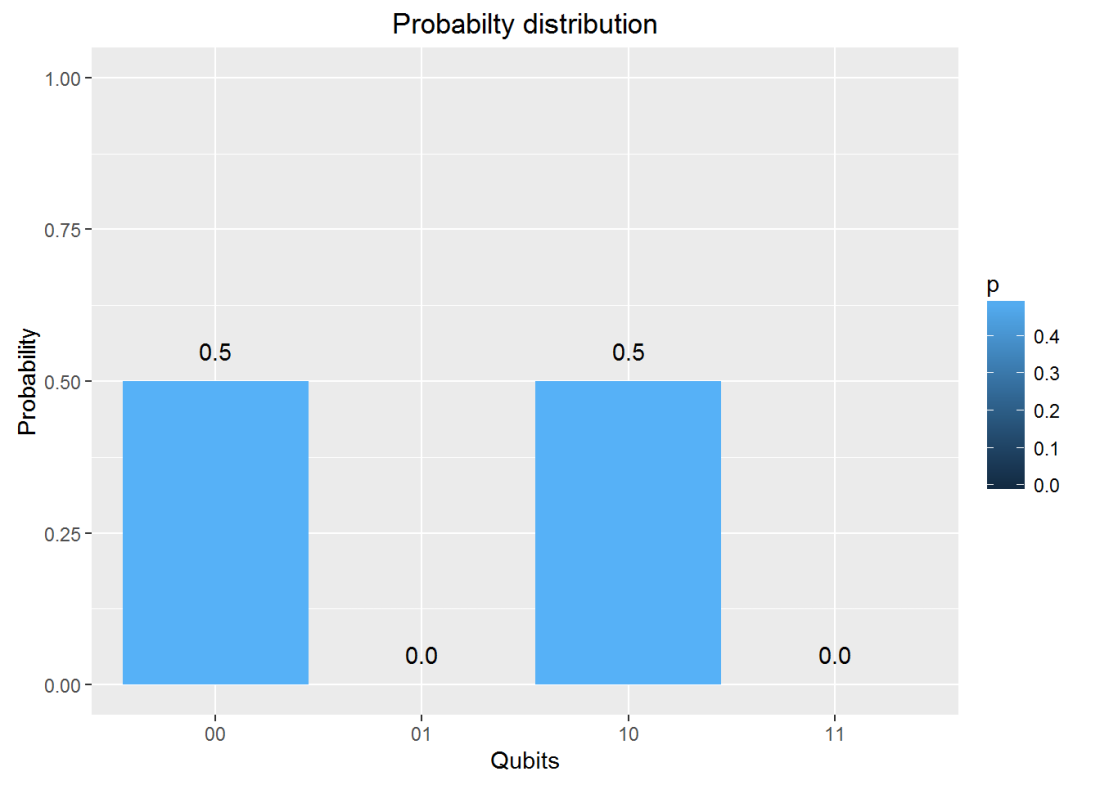
To obtain the results of gates in parallel we have to take the Tensor Product *Note:In the TensorProduct invocation the Identity matrix is passed as an argument to get the unitary matrix of the gate.* Q0 – Hadamard-Measurement Q1 – Identity- Measurement

#

a = TensorProd(Hadamard(I2),I2)

b = DotProduct(a,q00\_)

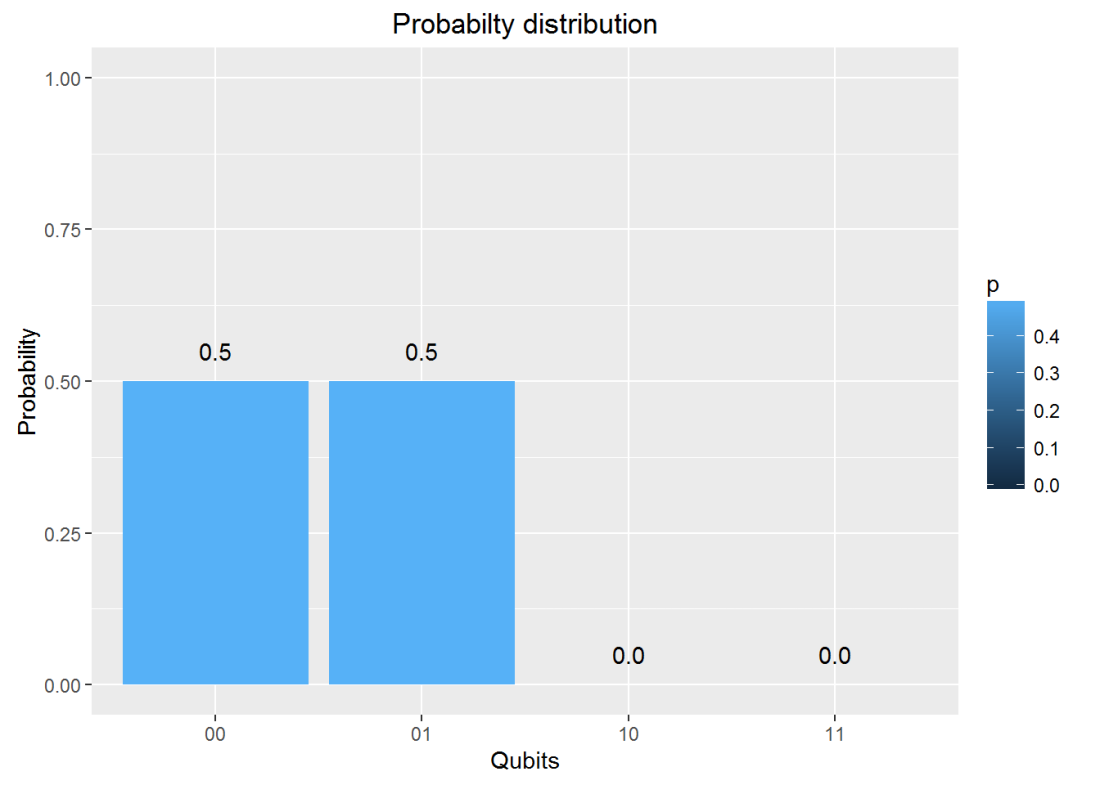
plotMeasurement(measurement(b))

[](https://gigadom.wordpress.com/2016/06/23/introducing-qcsimulator-a-5-qubit-quantum-computing-simulator-in-r/fig1-1/)

a = TensorProd(PauliZ(I2),Hadamard(I2))

b = DotProduct(a,q00\_)

plotMeasurement(measurement(b))

[](https://gigadom.wordpress.com/2016/06/23/introducing-qcsimulator-a-5-qubit-quantum-computing-simulator-in-r/fig1-2-2/)

**8. Measurement**

The measurement of a Quantum circuit can be obtained using the measurement function. Consider the following Quantum circuit  
Q0 – H-T-H-T-S-H-T-H-T-H-T-H-S-Measurement where H – Hadamard gate, T – T Gate and S- S Gate

a = SGate(Hadamard(TGate(Hadamard(TGate(Hadamard(TGate(Hadamard(SGate(TGate(Hadamard(TGate(Hadamard(I2)))))))))))))

measurement(a)

## 0 1

## v 0.890165 0.109835

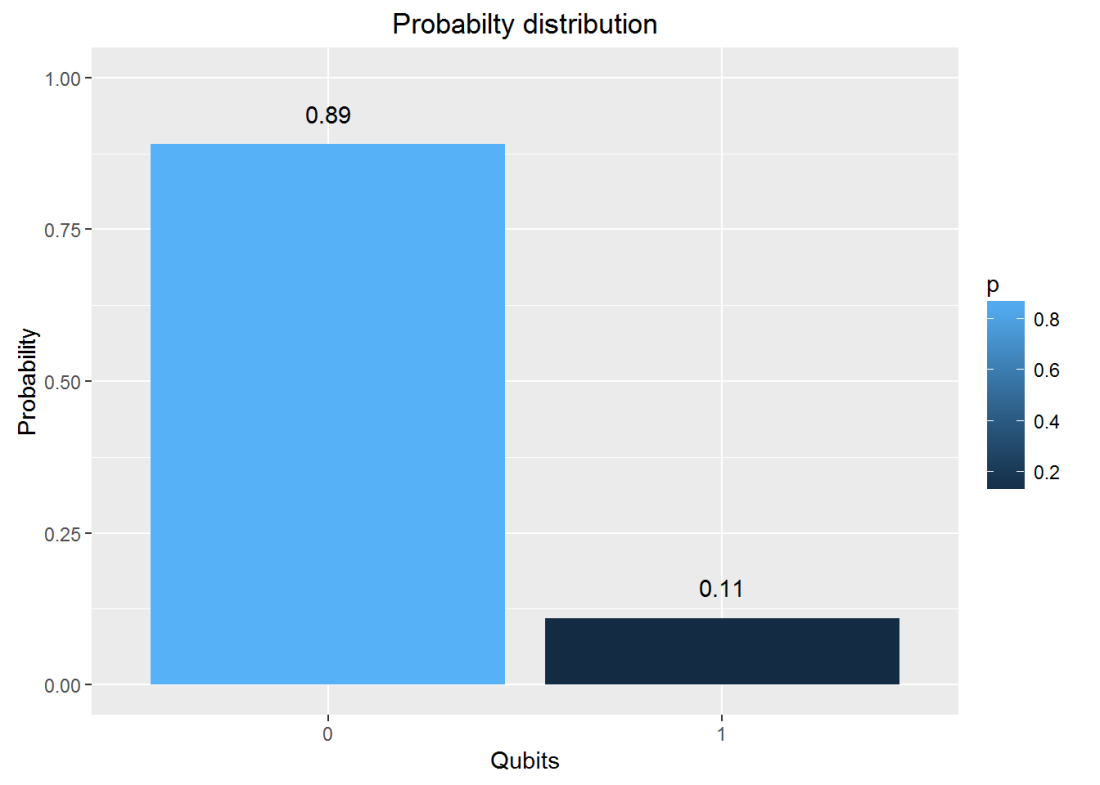
**9. Plot measurement**

Using the same example as above Q0 – H-T-H-T-S-H-T-H-T-H-T-H-S-Measurement where H – Hadamard gate, T – T Gate and S- S Gate we can plot the measurement

a = SGate(Hadamard(TGate(Hadamard(TGate(Hadamard(TGate(Hadamard(SGate(TGate(Hadamard(TGate(Hadamard(I2)))))))))))))

result = measurement(a)

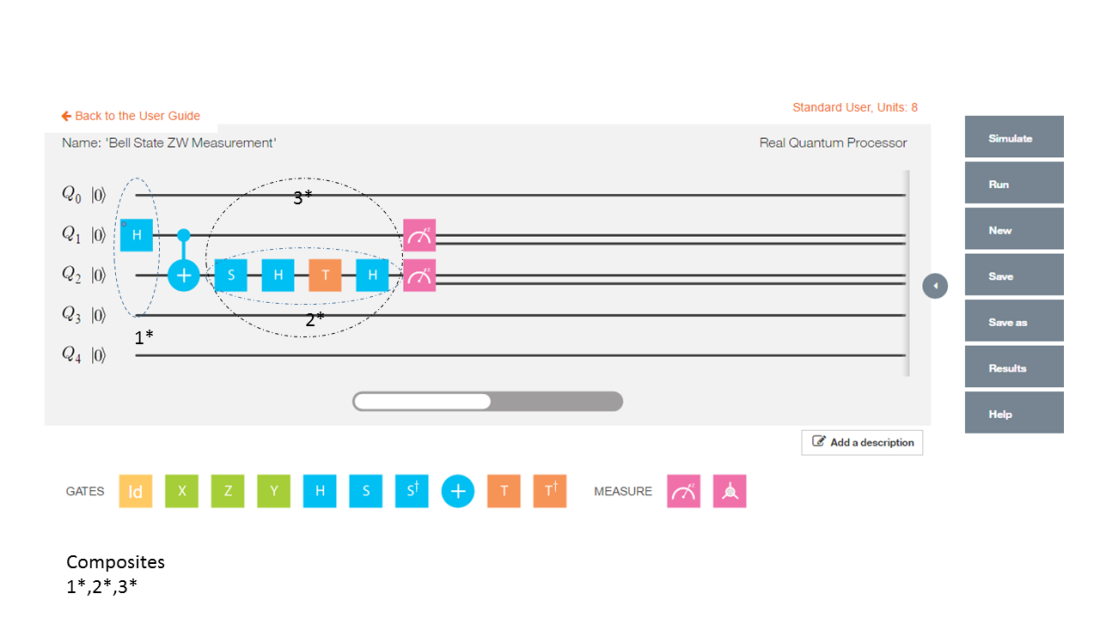
plotMeasurement(result)

[](https://gigadom.wordpress.com/2016/06/23/introducing-qcsimulator-a-5-qubit-quantum-computing-simulator-in-r/fig2-1/)

**10. Evaluating a Quantum Circuit**

The above procedures for evaluating a quantum gates in series and parallel can be used to evalute more complex quantum circuits where the quantum gates are in series and in parallel.

Here is an evaluation of one such circuit, the Bell ZQ state using the QCSimulator (from IBM’s Quantum Experience)

[](https://gigadom.wordpress.com/2016/06/23/introducing-qcsimulator-a-5-qubit-quantum-computing-simulator-in-r/pic3/)

# 1st composite

a = TensorProd(Hadamard(I2),I2)

# Output of CNOT

b = CNOT2\_01(a)

# 2nd series

c=Hadamard(TGate(Hadamard(SGate(I2))))

#3rd composite

d= TensorProd(I2,c)

# Output of 2nd composite

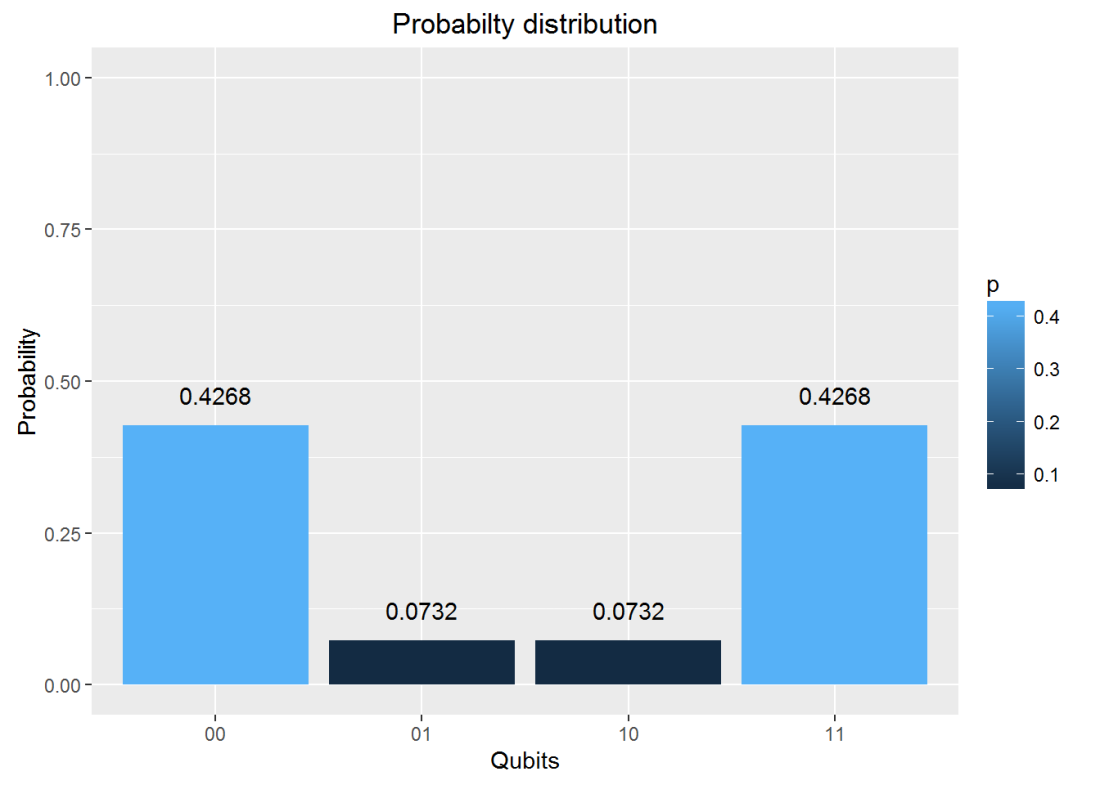
e = DotProduct(b,d)

#Action of quantum circuit on |00>

f = DotProduct(e,q00\_)

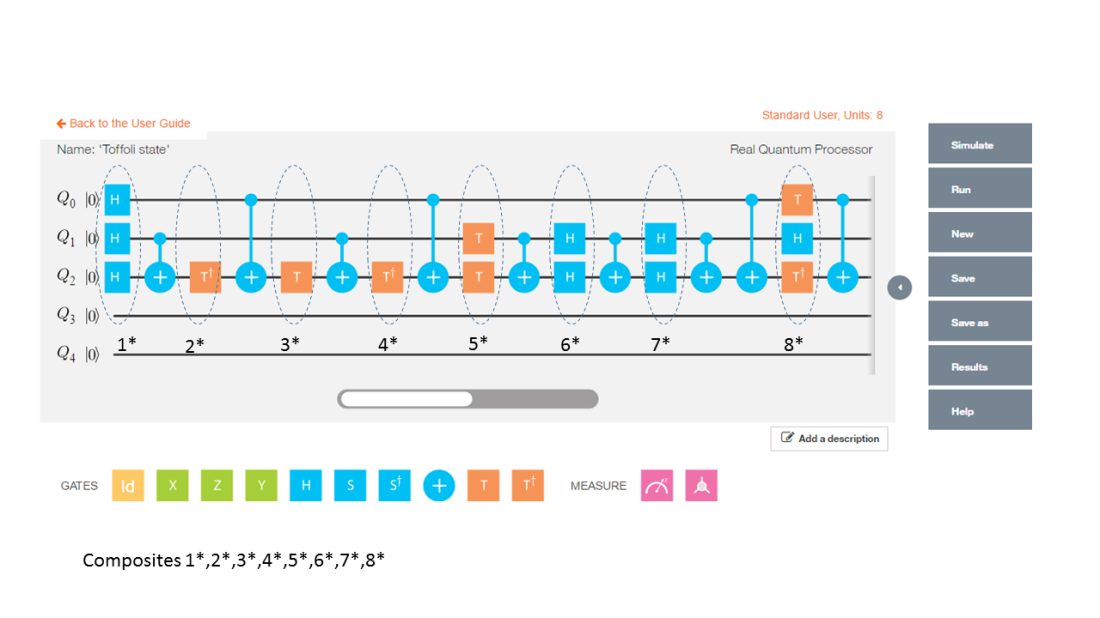
result= measurement(f)

plotMeasurement(result)

[](https://gigadom.wordpress.com/2016/06/23/introducing-qcsimulator-a-5-qubit-quantum-computing-simulator-in-r/fig3-1/)

**11. Toffoli State**

This circuit for this comes from IBM’s Quantum Experience. This circuit is available in the package. This is how the state was constructed. This circuit is shown below

[](https://gigadom.wordpress.com/2016/06/23/introducing-qcsimulator-a-5-qubit-quantum-computing-simulator-in-r/pic2-3/)

The implementation of the above circuit in QCSimulator is as below

# Computation of the Toffoli State

H=1/sqrt(2) \* matrix(c(1,1,1,-1),nrow=2,ncol=2)

I=matrix(c(1,0,0,1),nrow=2,ncol=2)

# 1st composite

# H x H x H

a = TensorProd(TensorProd(H,H),H)

# 1st CNOT

a1= CNOT3\_12(a)

# 2nd composite

# I x I x T1Gate

b = TensorProd(TensorProd(I,I),T1Gate(I))

b1 = DotProduct(b,a1)

c = CNOT3\_02(b1)

# 3rd composite

# I x I x TGate

d = TensorProd(TensorProd(I,I),TGate(I))

d1 = DotProduct(d,c)

e = CNOT3\_12(d1)

# 4th composite

# I x I x T1Gate

f = TensorProd(TensorProd(I,I),T1Gate(I))

f1 = DotProduct(f,e)

g = CNOT3\_02(f1)

#5th composite

# I x T x T

h = TensorProd(TensorProd(I,TGate(I)),TGate(I))

h1 = DotProduct(h,g)

i = CNOT3\_12(h1)

#6th composite

# I x H x H

j = TensorProd(TensorProd(I,Hadamard(I)),Hadamard(I))

j1 = DotProduct(j,i)

k = CNOT3\_12(j1)

# 7th composite

# I x H x H

l = TensorProd(TensorProd(I,Hadamard(I)),Hadamard(I))

l1 = DotProduct(l,k)

m = CNOT3\_12(l1)

n = CNOT3\_02(m)

#8th composite

# T x H x T1

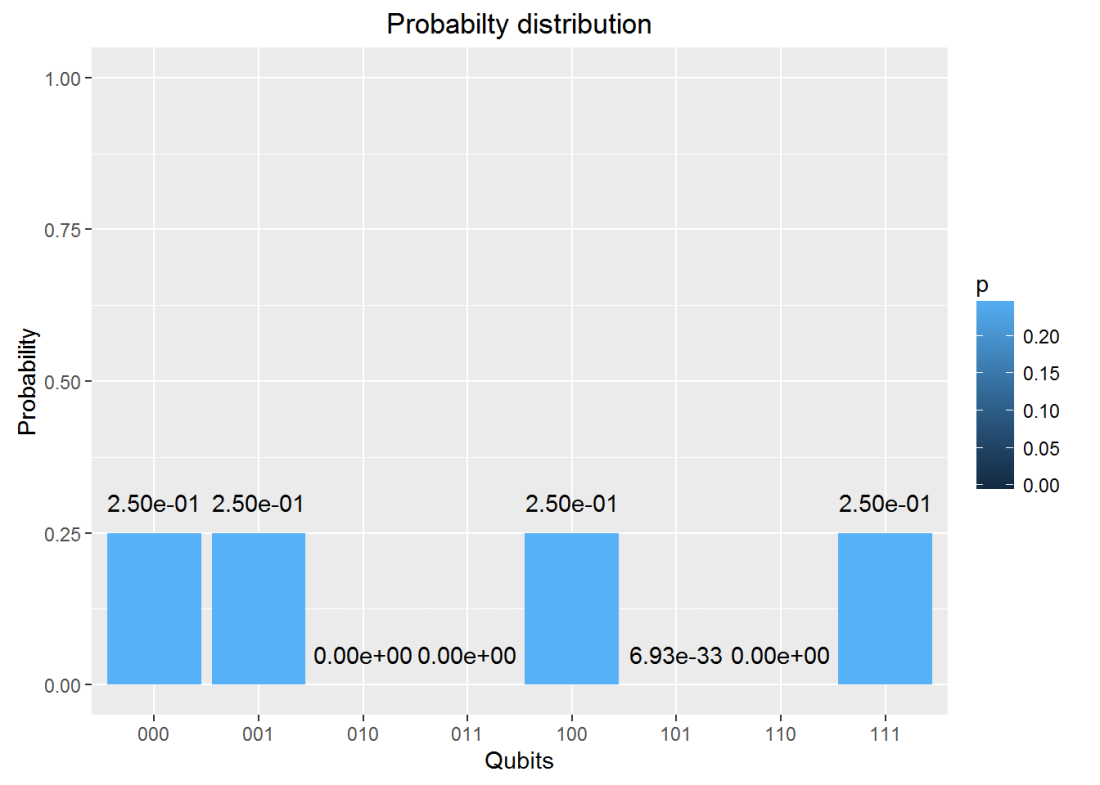
o = TensorProd(TensorProd(TGate(I),Hadamard(I)),T1Gate(I))

o1 = DotProduct(o,n)

p = CNOT3\_02(o1)

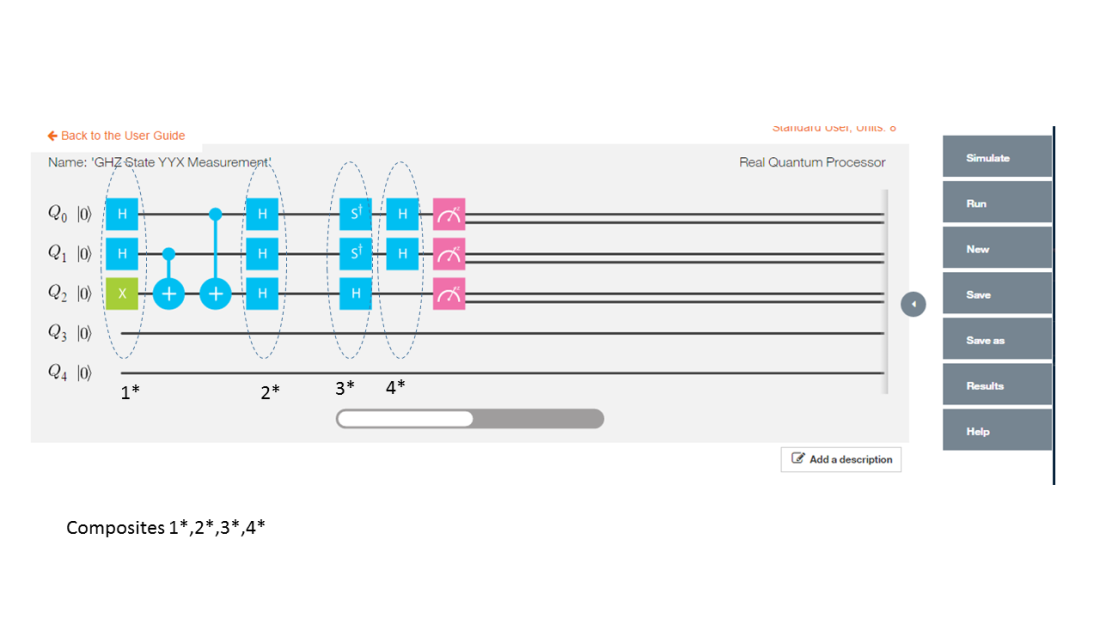
result = measurement(p)

plotMeasurement(result)

[](https://gigadom.wordpress.com/2016/06/23/introducing-qcsimulator-a-5-qubit-quantum-computing-simulator-in-r/fig4-1/)

**12. GHZ YYX measurement**

Here is another Quantum circuit, namely the entangled GHZ YYX state. This is

[](https://gigadom.wordpress.com/2016/06/23/introducing-qcsimulator-a-5-qubit-quantum-computing-simulator-in-r/pic1-2/)

and is implemented in QCSimulator as

# Composite 1

a = TensorProd(TensorProd(Hadamard(I2),Hadamard(I2)),PauliX(I2))

b= CNOT3\_12(a)

c= CNOT3\_02(b)

# Composite 2

d= TensorProd(TensorProd(Hadamard(I2),Hadamard(I2)),Hadamard(I2))

e= DotProduct(d,c)

#Composite 3

f= TensorProd(TensorProd(S1Gate(I2),S1Gate(I2)),Hadamard(I2))

g= DotProduct(f,e)

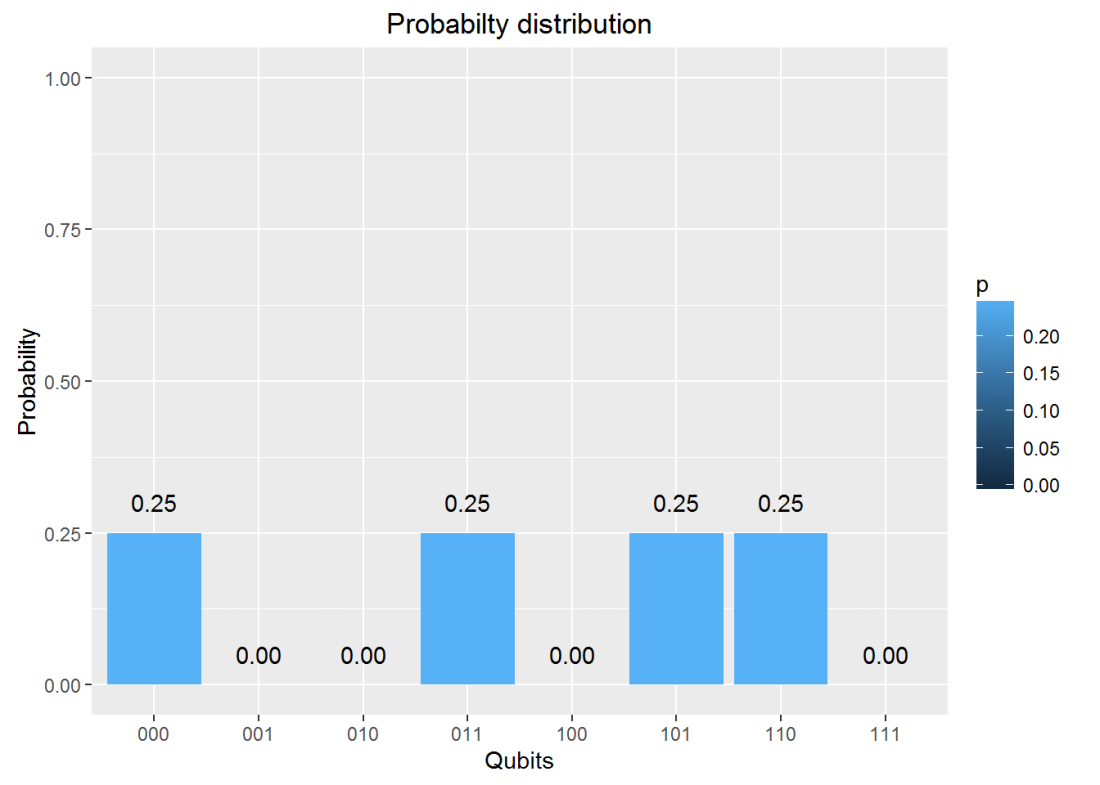
#Composite 4

i= TensorProd(TensorProd(Hadamard(I2),Hadamard(I2)),I2)

j = DotProduct(i,g)

result=measurement(j)

plotMeasurement(result)

[](https://gigadom.wordpress.com/2016/06/23/introducing-qcsimulator-a-5-qubit-quantum-computing-simulator-in-r/fig5-1/)