```
using Yao, YaoPlots
```

Applications

We'll discuss a few applications of qubits and quantum circuits. Namely the Bell States, Superdense Coding and Quantum Teleportation.

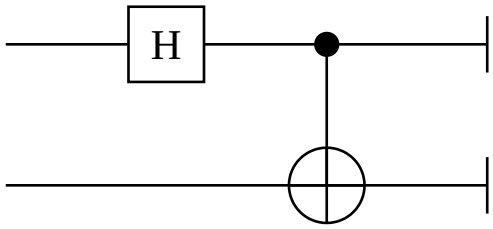
Bell States

The states,

$\frac{|00\rangle+|11\rangle}{\sqrt{2}}$, $\frac{|01\rangle+|10\rangle}{\sqrt{2}}$

$\frac{|00\rangle-|11\rangle}{\sqrt{2}}$ and $\frac{|01\rangle-|10\rangle}{\sqrt{2}}$

are known as the bell states. They are made by the bell circuit. The bell circuit looks like this.




```
begin
    bellcircuit = chain(2, put(1=>H), control(1, 2=>X))
    plot(bellcircuit)
end
```

As you can see, the circuit takes in two qubits as input, and operates on them to give the bell states.

Feeding qubits to a circuit in Yao

There are many ways to create qubits to feed to quantum circuits in Yao.

```
q1 = ArrayReg{1, Complex{Float64}, Array...}
        active qubits: 2/2
```

```
q1 = ArrayReg(bit"00") #creating the system of two qubits with state |00>.
```

```
4×1 Array{Complex{Float64},2}:
```

```
1.0 + 0.0im
0.0 + 0.0im
0.0 + 0.0im
0.0 + 0.0im
```

```
state(q1) #state of a qubit in vector form
```

Note: ***Normalizing*** *basically means making the summation of squares of probability amplitudes, equal to 1.*

Other ways of doing it are

```
ArrayReg{1, Complex{Float64}, Array...}
    active qubits: 2/2
```

```
ArrayReg(bit"00") + ArrayReg(bit"11") |> normalize! #Equivalent to (1/√2)*(|00>+|11>)
```

```
ArrayReg{1, Complex{Float64}, Array...}
    active qubits: 2/2
```

```
zero_state(2) #2 qubits, both with the state |0> and |0>
```

There! We have a system of two qubits! Let's try feeding the qubits to the Bell circuit we made!

```
a = ArrayReg{1, Complex{Float64}, Array...}
        active qubits: 2/2
```

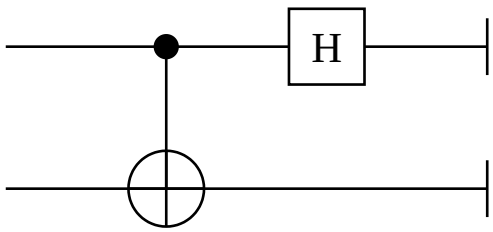
```
a = (q1 |> bellcircuit) #Passing the qubit q1 through the bell circuit
```

```
4×1 Array{Complex{Float64},2}:
 0.7071067811865475 + 0.0im
                0.0 + 0.0im
                0.0 + 0.0im
 0.7071067811865475 + 0.0im
```

```
state(a)
```

Reverse Bell Circuit

A circuit which reverses the effects of the bell circuit on a qubit. It's represented in a circuit as follows




```
begin
    reversebellcircuit = chain(2, control(1,2=>X), put(1=>H))
    plot(reversebellcircuit)
end
```

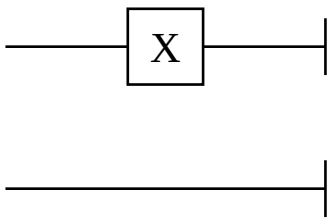
You can input the output state you got from the bell circuit, into the reverse bell circuit, and you'll get back your original state. Why not give it a try?

Let's pass the qubits we got by passing two qubits in the Bell circuit, into the Reverse Bell Circuit.

```
4×1 Array{Complex{Float64},2}:
 0.9999999999999998 + 0.0im
                0.0 + 0.0im
                0.0 + 0.0im
                0.0 + 0.0im
```

```
let
        res = (a |> reversebellcircuit)
        state(res)
end
```

What do you think is the effect of single qubit gates on a qubit, which is entangled with another qubit? Lets check it out!




```
begin
    singlequbitcircuit = chain(2, put(1=>X))
    plot(singlequbitcircuit)
end
```

```
4×1 Array{Complex{Float64},2}:
                0.0 + 0.0im
 0.7071067811865475 + 0.0im
 0.7071067811865475 + 0.0im
                0.0 + 0.0im
```

```
begin
    bellstate = ArrayReg(bit"00") + ArrayReg(bit"11") |> normalize!
    re = bellstate |> singlequbitcircuit
    state(re)
end
```

Can you notice how the circuit behaves as if its operating on one qubit? If the input is $|\{0\}[1]\rangle + |\{1\}[1]\rangle$, it behaves like the bits in the curly braces are the state of one qubit, while the bits in sqare brackets are the state of the other qubit, and it operates on them accordingly?

Super-dense Coding and Quantum Teleportation

Suppose we have two entangled qubits in a bell state, represented by $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$. Alice and Bob are two friends. Alice gets one qubit and Bob gets the other one. Both of them travel far apart with their

qubits, and not measuring their qubits so as to preserve the entangled state. The above information is the premise for both:

1. Super-dense coding
2. Quantum Teleportation

1. Super-dense coding

Alice wants to send Bob two classical bits of information. That means, one of the four states $00, 01, 10$ and 11. How does she do this? Lets say she achieves this by passing her qubits through one of the following gates, corresponding to the information she wants to send.

1. If she wants to send Bob 00, then she'll send her qubit to Bob, as it is
2. If she wants to send Bob 01, then she'll send her qubit to Bob, after passing it through X gate.
3. If she wants to send Bob 10, then she'll send her qubit to Bob, after passing it through Z gate.
4. If she wants to send Bob 11, then she'll send her qubit to Bob, after passing it through Y gate.

Bob will then run both the qubits through the reverse bell circuit and measure them and he'll get what Alice wanted to send him.

Measuring qubits in Yao can be done by using the measure function.

Assume you've the a qubit or a system of qubits in q, and you want to measure it, the syntax is,

```
q |> r->measure(r, nshots=number_of_runs)
```

We can't determine the probability amplitudes of a qubit. But, we can run the measurement many times, and count the frequency of each bit combination it gives.

If you don't get that, just know the syntax and keep nshots=1024.

Now, lets try measuring the result of the previous assignment!

```
measuredqubits =
▸ BitBasis.BitStr{2,Int64}[01 ₍₂₎, 10 ₍₂₎, 01 ₍₂₎, 10 ₍₂₎, 01 ₍₂₎, 01 ₍₂₎, 10 ₍₂₎, 01
```

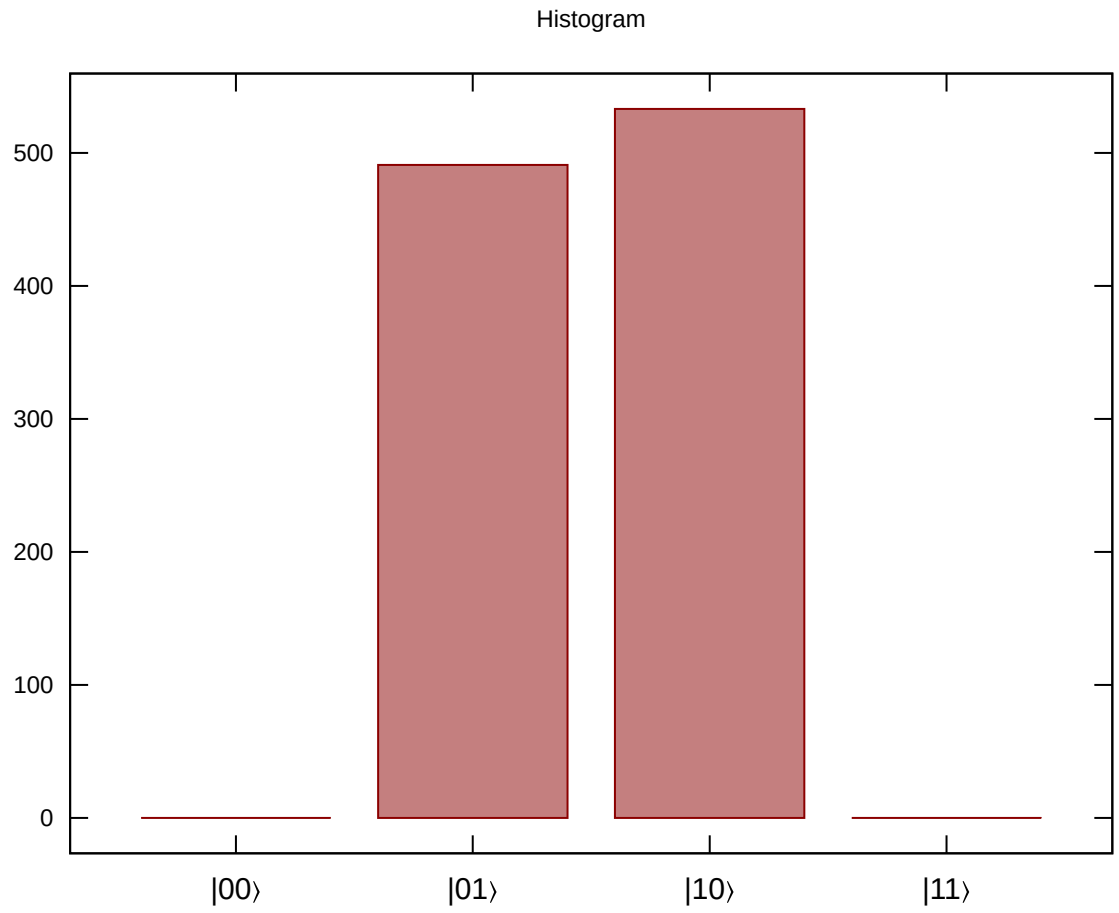
```
measuredqubits = re |> r->measure(r, nshots=1024)
```

Sometimes the measurement gives $|01\rangle$ and sometimes $|10\rangle$. The probability of getting a $|01\rangle$ is same as getting the probability of $|10\rangle$, which is $(1/\sqrt{2})^2$ that is 0.5 .

Wanna see how we can test that? The below function plots the probability as histograms. You don't need to know its inner workings to use it.

```
plotmeasure (generic function with 1 method)
```

```
begin
    using BitBasis: BitStr
    using StatsBase: fit, Histogram
    using Gaston: bar, set, Axes
    set(showable="svg")
    function plotmeasure(x::Array{BitStr{n,Int},1}) where n
        set(preamble="set xtics font ',$(n<=3 ? 15 : 15/(2^(n-3)))'")
        hist = fit(Histogram, Int.(x), 0:2^n)
        bar(hist.edges[1][1:end-1], hist.weights, fc="'dark-red'", Axes(title =
:Histogram, xtics = (0:(2^n-1), "|" .* string.(0:(2^n-1), base=2, pad=n) .* "⟩")))
    end
end
```




```
plotmeasure(measuredqubits)
```

The probability of the measurement giving $|01\rangle$ is 47.94921875% and the number of times the measurement result is $|10\rangle$ is 52.05078125%.

Implementing the superdense coding.

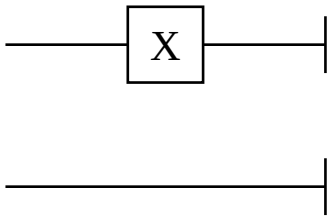
```
"01"
```

```
begin
    Alice_and_Bobs_entangled_qubits = ArrayReg(bit"00") + ArrayReg(bit"11") |>
normalize!
    input = ["00" "01" "10" "11"][rand(1:4)] #Assuming Alice wants to send one of
these inputs to Bob.
```

```
end
```

```
4×1 Array{Complex{Float64},2}:
 0.7071067811865475 + 0.0im
                0.0 + 0.0im
                0.0 + 0.0im
 0.7071067811865475 + 0.0im
```

```
state(Alice_and_Bobs_entangled_qubits)
```




```
begin
    if(input == "00")
        Alices_circuit = chain(2)
    elseif(input == "01")
        Alices_circuit = chain(2, put(1=>X))
    elseif(input == "10")
        Alices_circuit = chain(2, put(1=>Z))
    elseif(input == "11")
        Alices_circuit = chain(2, put(1=>Y))
    end
    plot(Alices_circuit)
end
```

```
Bobs_part =
```

```
▸ BitBasis.BitStr{2,Int64}[10 ₍₂₎, 10 ₍₂₎, 10 ₍₂₎, 10 ₍₂₎, 10 ₍₂₎, 10 ₍₂₎, 10 ₍₂₎, 10
```

```
Bobs_part = ((Alice_and_Bobs_entangled_qubits |> Alices_circuit) |>
reversebellcircuit) |> r->measure(r, nshots=1024)
#The content in the first round bracket, outputs qubits, which are then fed to
reversebellcircuit we saw before, which is then fed to the measure function.
```

2. Quantum Teleportation

Alice now has one more qubit, in the state $a|0\rangle + b|1\rangle$. She wants to send her qubit to Bob, by changing the state of Bob's entangled qubit to Alice's extra qubit.

Confusing? Let's try naming the qubits. Alice and Bob have the qubits A and B respectively, which are entangled in the bell state. Alice has another qubit C.

Alice wants to send her qubit to Bob by changing the state of B, to that of C. She wants the state of the qubit B to be $a|0\rangle + b|1\rangle$. Alice doesn't know the values of a and b. Also, Bob can't make a measurement in any case as doing so will destroy B's state.

How does Alice do it? Well, she first passes both her qubits through the Reverse Bell circuit. She then measures both of her qubits. She gets one of the following outputs: $|00\rangle, |01\rangle, |10\rangle or |11\rangle$, each with probability 1/4. Alice sends these two classical bits to Bob, via any classical means, example, she texts him or calls and tells him. Bob knows that corresponding to her message, there are 4 options, and he takes action accordingly for each.

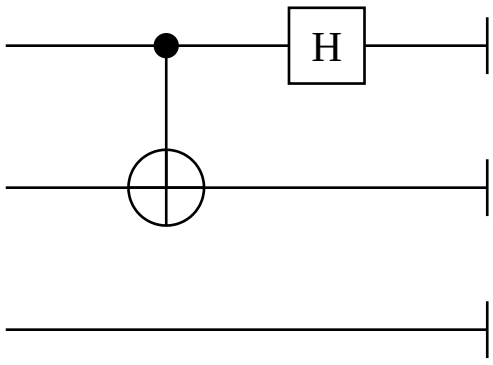
1. He gets the message, $|00\rangle$, from Alice and knows that his qubit(qubit B) changed its state to that of qubit C.
2. He gets the message, $|01\rangle$, and knows that his qubit changed its state to $a|1\rangle + b|0\rangle$, and he passes his qubit through X gate to change its state to that of qubit C.
3. He gets the message, $|10\rangle$, and knows that his qubit changed its state to $a|0\rangle - b|1\rangle$, and he passes his qubit through Z gate to change its state to that of qubit C.
4. He gets the message, $|11\rangle$, and knows that his qubit changed its state to $a|1\rangle - b|0\rangle$, and he passes his qubit through Y gate to change its state to that of qubit C.

```
2×1 Array{Complex{Float64},2}:
 0.44489907827390895 - 0.82452279055161461im
 -0.2056595065687468 + 0.28272096733334784im
```

```
begin
    Alices_and_Bobs_entangled_qubits = ArrayReg(bit"00") + ArrayReg(bit"11") |> normalize!
    Alicequbit = rand_state(1) #This function creates a qubit with a random state.
    state(Alicequbit)
end
```

```
4×1 Array{Complex{Float64},2}:
 0.7071067811865475 + 0.0im
                0.0 + 0.0im
                0.0 + 0.0im
 0.7071067811865475 + 0.0im
```

```
state(Alices_and_Bobs_entangled_qubits)
```




```
begin
    teleportationcircuit = chain(3, control(1,2=>X), put(1=>H))
    plot(teleportationcircuit)
end
```

```
8×1 Array{Complex{Float64},2}:
  0.22244953913695442 - 0.41226139527580724im
  0.22244953913695442 - 0.41226139527580724im
 -0.10282975328437338 + 0.14136048366667 39im
  0.10282975328437338 - 0.14136048366667 39im
 -0.10282975328437338 + 0.14136048366667 39im
  0.10282975328437338 - 0.14136048366667 39im
  0.22244953913695442 - 0.41226139527580724im
  0.22244953913695442 - 0.41226139527580724im
```

```
begin
    feeding = join(Alices_and_Bobs_entangled_qubits, Alicequbit) |> teleportationcircuit
    state(feeding)
end
```

The `join(qubit1, qubit2.....,qubitn)` function is used to join multiple qubits. Remember, the circuit takes the qubits as inputs, in reverse order.

```
Alices_measuredqubits = 00 ₍₂₎
```

```
Alices_measuredqubits = measure_remove!(feeding, 1:2)
```

```
ArrayReg{1, Complex{Float64}, Array...}
    active qubits: 1/1
```

```
if(Alices_measuredqubits == bit"00")
    Bobs_qubit = feeding
elseif(Alices_measuredqubits == bit"01")
    Bobs_qubit = feeding |> chain(1, put(1=>Z))
elseif(Alices_measuredqubits == bit"10")
    Bobs_qubit = feeding |> chain(1, put(1=>X))
else
    Bobs_qubit = feeding |> chain(1, put(1=>Y))
end
```

The difference between `measure(input, location of qubits(optional, measures all qubits if absent), nshots=number of runs)` and `measure_remove!(input, the location of qubits(optional, measures all qubits if absent))` is that the latter collapses the state of the qubit, and removes the $0 + 0im$ rows of the state vector, while the former only measures the state of the qubit.

```
2×1 Array{Complex{Float64},2}:
    0.444899078273909 - 0.8245227905516147im
 -0.20565950656874682 + 0.282720967333479im
```

```
state(Bobs_qubit)
```

Is Alice's qubit same as Bob's qubit now? You can see that for yourself!

```
2×2 Array{Complex{Float64},2}:
 0.444899-0.824523im  0.444899-0.824523im
 -0.20566+0.282721im  -0.20566+0.282721im
```

```
[state(Alicequbit) state(Bobs_qubit)]
```

Left side : State of Alice's qubit. Right side : State of Bob's qubit. Almost equivalent!