More Quantum Gates

As you might have thought, there do exist gates, other than the previously defined ones.

Single qubit gates

The R_{φ}^{Z} gate

Passing a qubit through the R_{φ}^{Z} is equivalent to multiplying its state vector by $\begin{bmatrix} 1 & 0 \\ 0 & e^{\varphi i} \end{bmatrix}$. Remember, $e^{i\theta} = cos(\theta) + isin(\theta)$.

The R_{φ}^{Z} gate can be alternatively denoted by, $\begin{bmatrix} e^{-\varphi i/2} & 0 \\ 0 & e^{\varphi i/2} \end{bmatrix}$. Its just the original matrix, multiplied by $e^{-\varphi i/2}$. We can do this since multiplication by $e^{-\varphi i/2}$ is not *observable* during measurement as its a complex unit and $|e^{i\theta}| = |cos(\theta) + isin(\theta)| = 1$. Remember that the abstract value of a complex number $a + ib$, i.e., $|a + ib| = \sqrt{a^2 + b^2}$ and $sin^2\theta + cos^2\theta = 1$.

Considering a qubit, $a|0\rangle + b|0\rangle$, passing it through the $R_{\frac{\pi}{2}}^{Z}$ gate is equivalent to $\begin{bmatrix} 1 & 0 \\ 0 & e^{\pi i/2} \end{bmatrix}$. And since $cos(\frac{\pi}{2}) = 0$ and $sin(\frac{\pi}{2}) = 1$, we can rewrite the above as, $\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$.

Lets try the above in Yao! The R_{φ}^{Z} gate can be used in Yao with the shift *block*.

```
using Yao, YaoPlots
```

```
2×1 Array{Complex{Float64},2}:
   0.6924477114839013 - 0.6829633197845618im
 -0.10091113544888164 - 0.20950945905131718im
```

```
begin
    qubit = rand_state(1)
    state(qubit)
end
```

```
2×1 Array{Complex{Float64},2}:
  0.6924477114839013 - 0.6829633197845618im
 0.20950945905131718 - 0.10091113544888165im
```

```
state(qubit |> chain(1, put(1=>shift(π/2))))
```

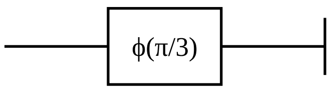
As expected the output was $\begin{bmatrix} a \\ ib \end{bmatrix}$. Remember, $i = \sqrt{-1}$ and $i^2 = -1$. (Also, note that in Julia, imaginary number i is represented by im.)

Also, R^Z_π gate is equivalent to Z gate.

```
true
```

```julia
round.(Matrix(chain(1, put(1=>shift(π))))) == round.(Matrix(chain(1, put(1=>Z))))
#The round functions "rounds-off" the elements of the matrices
```

Its represented in a circuit diagram by,



The T Gate

The T gate is equivalent to $R^Z_{\frac{\pi}{4}}$. In its matrix form, it can be written as $\begin{bmatrix} 1 & 0 \\ 0 & \frac{1+i}{\sqrt{2}} \end{bmatrix}$. Nevertheless, in Yao, it can be used by using the **T** *block* .

```
2×1 Array{Complex{Float64},2}:
 0.6924477114839013 - 0.6829633197845618im
 0.2195005073910501 + 0.07679061104477325im
```

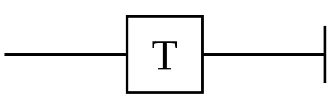
```julia
state(qubit |> chain(1, put(1=>T)))
```

Also,

```
true
```

```julia
Matrix(chain(1, put(1=>shift(π/4)))) == Matrix(chain(1, put(1=>T)))
```

Its circuit diagram representation looks somewhat like -



The R^X_φ gate

Similar to the R^Z_φ gate, the R^X_φ gate can be represented by $\begin{bmatrix} cos(\frac{\varphi}{2}) & -sin(\frac{\varphi}{2})i \\ -sin(\frac{\varphi}{2})i & cos(\frac{\varphi}{2}) \end{bmatrix}$.

The R^Y_φ gate

Similar to the R^Z_φ gate, the R^X_φ gate can be represented by $\begin{bmatrix} cos(\frac{\varphi}{2}) & -sin(\frac{\varphi}{2}) \\ sin(\frac{\varphi}{2}) & cos(\frac{\varphi}{2}) \end{bmatrix}$.

They can be represented in Yao using the **Rx** and **Ry** *blocks* respectively

```
2×1 Array{Complex{Float64},2}:
 0.07679061104477329 - 0.21950050739105015im
 -0.6829633197845618 - 0.6924477114839013im
```

```
state(qubit |> chain(1, put(1=>Rx(π))))
```

```
2×1 Array{Complex{Float64},2}:
  0.6829633197845618 + 0.6924477114839013im
 0.07679061104477325 - 0.2195005073910502im
```

```
state(qubit |> chain(1, put(1=>Ry(π))))
```

There's also an **Rz** *block* which which represents the alternative form of R^Z_φ matrix, i.e., $\begin{bmatrix} e^{-\varphi i/2} & 0 \\ 0 & e^{\varphi i/2} \end{bmatrix}$.

```
2×1 Array{Complex{Float64},2}:
 0.6924477114839013 - 0.6829633197845618im
 0.2195005073910502 + 0.07679061104477324im
```

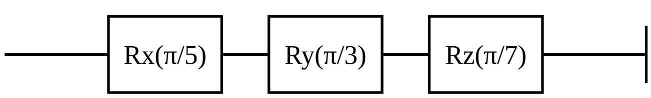
```
state(qubit |> chain(1, put(1=>Rz(π))))
```

Note that the absolute value of both the shift and Rz blocks are same.

```
true
```

```
abs.(Matrix(chain(1, put(1=>shift(π/5))))) == abs.(Matrix(chain(1, put(1=>Rz(π/5)))))
```

The circuit diagram representations of Rx, Ry and Rz blocks, respectively



Multi-qubit Gates

The SWAP Gate

The SWAP gate swaps the state of two qubits. It can be represented by the matrix, $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

Its represented in Yao via the **swap** *block*.

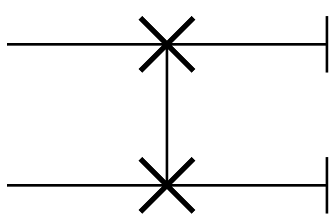
```
4×1 Array{Complex{Float64},2}:
  0.37506700539820803 + 0.5151843025868749im
   0.6485827649421082 + 0.19688812148771873im
  -0.1304450041897797 - 0.19463159785685502im
 -0.17552844135986223 + 0.22085685123666945im
```

```
begin
    q = rand_state(2)
    state(q)
end
```

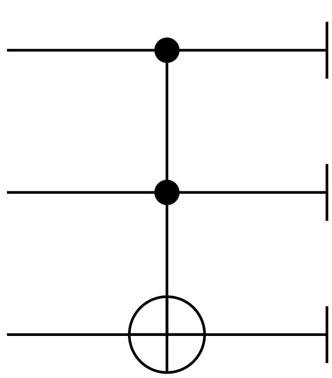
```
4×1 Array{Complex{Float64},2}:
  0.37506700539820803 + 0.5151843025868749im
  -0.1304450041897797 - 0.19463159785685502im
   0.6485827649421082 + 0.19688812148771873im
 -0.17552844135986223 + 0.22085685123666945im
```

```
state(q |> chain(2, swap(1,2)))
```

The SWAP gate has the following circuit diagram representation

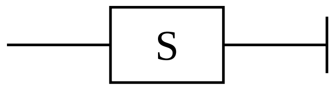


There's a Toffoli gate, an S gate, a CSWAP gate, a $CR_{\varphi}^{X,Y,Z}$ gate and probably a lot more. They can all be constructed using the existing blocks in Yao.

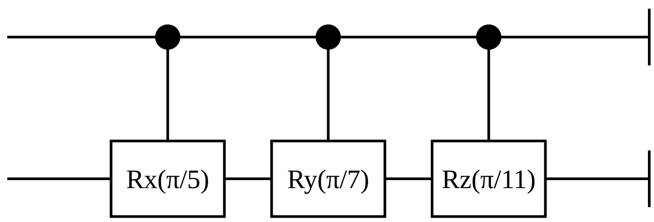


```
let
    toffoli_Gate = chain(3, control(1:2, 3=>X)) #The toffoli gate
    plot(toffoli_Gate)
```

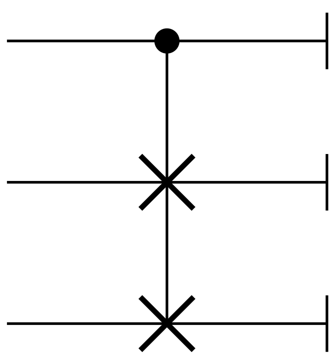
```
end
```




```
let
    S_Gate = chain(1, put(1 => label(shift(π/2), "S"))) #The S gate
    plot(S_Gate)
end
```



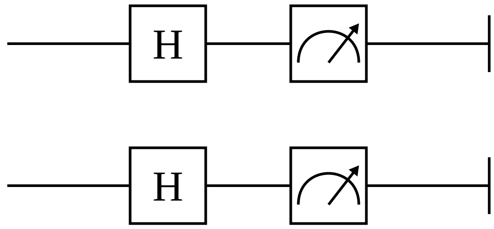

```
let
    CRXYZ_Gates = chain(2, control(1, 2=>Rx(π/5)), control(1, 2=>Ry(π/7)), control(1,
2=>Rz(π/11)))
    plot(CRXYZ_Gates)
end
```



```
let
    CSWAP = chain(3, control(1, 2:3=>SWAP)) #The CSWAP gate
    plot(CSWAP)
end
```

The Measure Gate

We already know how to measure the qubits. We can do it in the circuit itself too.




```julia
begin
    MeasureGate = chain(2, repeat(H, 1:2), Measure(2, locs=1:2))
    plot(MeasureGate)
end
```

Note that now, when we measure them using the measure block, the output remains unchanged, even though we should've a 25% chance of getting $|00\rangle$, $|01\rangle$, $|10\rangle$ or $|11\rangle$.

```
▸BitBasis.BitStr{2,Int64}[00 (2), 00 (2), 00 (2), 00 (2), 00 (2), 00 (2), 00 (2), 00
```

```julia
zero_state(2) |> MeasureGate |> r->measure(r, nshots=1024)
```

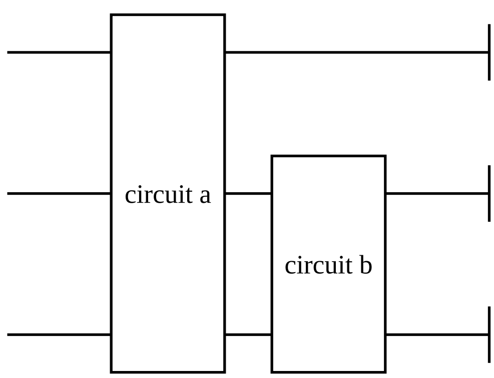
Without the Measure gate,

```
▸BitBasis.BitStr{2,Int64}[01 (2), 00 (2), 10 (2), 11 (2), 11 (2), 10 (2), 10 (2), 10
```

```julia
zero_state(2) |> repeat(2, H, 1:2) |> r->measure(r, nshots=1024)
```

The LabeledBlock

Its used for easily plotting circuits as boxes for simpler visualization.

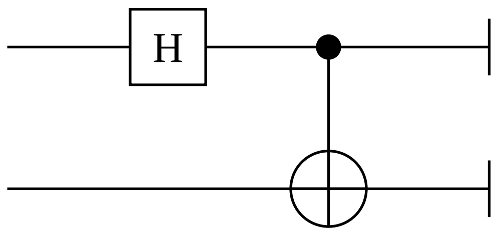



```julia
let
    a = chain(3, repeat(H, 1:2), put(3=>X))
    b = chain(2, repeat(Y, 1:2))
    circuit = chain(3, put(1:3 => label(a, "circuit a")), put(2:3 => label(b,
"circuit b")))
    plot(circuit)
end
```

Daggered Block

We use Daggered block to build circuits which undo the effects of a particular circuit.

Let's take an example. Remember Bell Circuit?




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

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

```
state(zero_state(2) |> bellcircuit)
```

Building the Reverse Bell Circuit is as easy as,

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

```
state(zero_state(2) |> bellcircuit |> Daggered(bellcircuit))
```

Plotting the Daggered Block is a bit tricky! YaoPlots doesn't support DaggeredBlock yet. There are two alternatives to this.

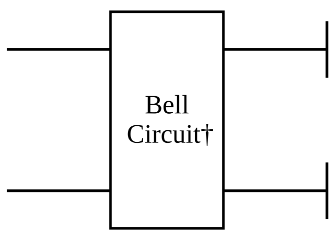
```
block type YaoBlocks.Daggered{YaoBlocks.ChainBlock{2},2} does not support visualization.

  1. error(::String) @ error.jl:33
  2. draw!(::YaoPlots.CircuitGrid, ::YaoBlocks.Daggered{YaoBlocks.ChainBlock{2},2},
     ::Array{Int64,1}, ::Array{Any,1}) @ vizcircuit.jl:138
  3. >>(::YaoBlocks.Daggered{YaoBlocks.ChainBlock{2},2},
     ::YaoPlots.CircuitGrid) @ vizcircuit.jl:267
  4. (::YaoPlots.var"#23#24"{YaoBlocks.Daggered{YaoBlocks.ChainBlock{2},2}})
     (::YaoPlots.CircuitGrid) @ vizcircuit.jl:252
  5. (::YaoPlots.var"#26#27"{YaoPlots.var"#23#24"
     {YaoBlocks.Daggered{YaoBlocks.ChainBlock{2},2}},YaoPlots.CircuitGrid})
     () @ vizcircuit.jl:259
  6. canvas(::YaoPlots.var"#26#27"{YaoPlots.var"#23#24"
     {YaoBlocks.Daggered{YaoBlocks.ChainBlock{2},2}},YaoPlots.CircuitGrid}) @ brush.jl:1
```

```
 7. #circuit_canvas#25(::Float64, ::Float64, ::typeof(YaoPlots.circuit_canvas),
    ::YaoPlots.var"#23#24"{YaoBlocks.Daggered{YaoBlocks.ChainBlock{2},2}},
    ::Int64) @ vizcircuit.jl:258
 8. #vizcircuit#22(::Float64, ::Float64, ::Float64, ::typeof(YaoPlots.vizcircuit),
    ::YaoBlocks.Daggered{YaoBlocks.ChainBlock{2},2}) @ vizcircuit.jl:251
 9. vizcircuit(::YaoBlocks.Daggered{YaoBlocks.ChainBlock{2},2}) @ vizcircuit.jl:251
10. #plot#21(::Base.Iterators.Pairs{Union{},Union{},Tuple{},NamedTuple{(),Tuple{}}},
    ::typeof(YaoPlots.plot),
    ::YaoBlocks.Daggered{YaoBlocks.ChainBlock{2},2}) @ vizcircuit.jl:249
11. plot(::YaoBlocks.Daggered{YaoBlocks.ChainBlock{2},2}) @ vizcircuit.jl:249
12. top-level scope @ Local: 1  [inlined]
```

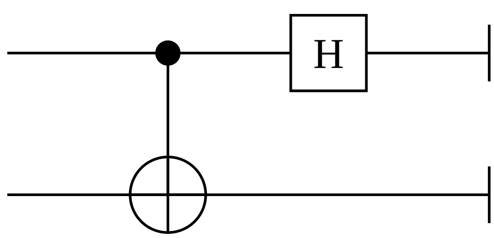
```
plot(Daggered(bellcircuit))
```

One way is to use the label block




```
let
    reversebellcircuit = put(2, 1:2 => label(Daggered(bellcircuit), "Bell\n
Circuit†"))
    plot(reversebellcircuit)
end
```

Another way, is to use `'`.



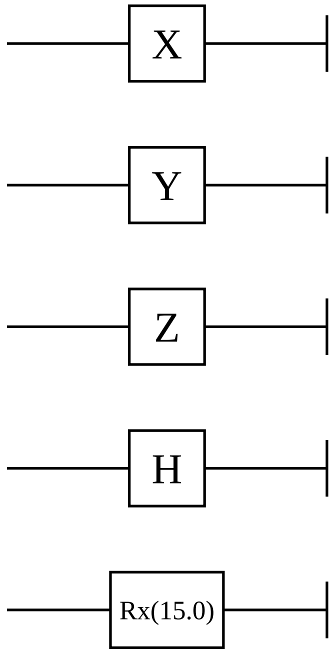

```
plot(bellcircuit')
```

Whats the difference between the DaggeredBlock and adjoint `'` ?

One is a function of Yao, while another of Base julia. Both perform the same operation on their input, but DaggeredBlock is many times faster.

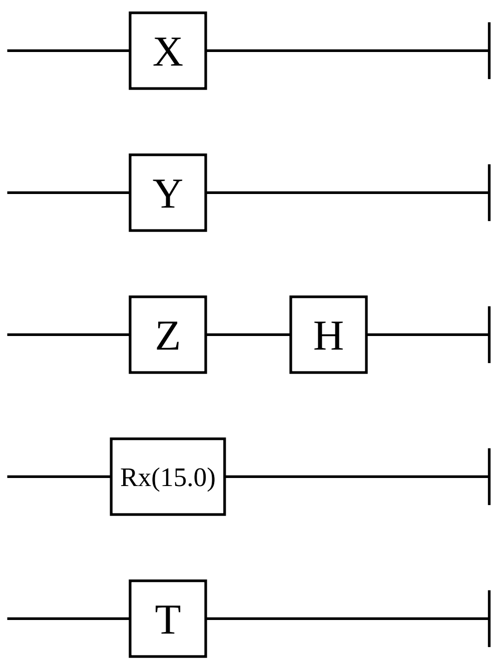
The Kron Block

Consider you've to make the below circuit.



Tired of using `put` block after `put` block?

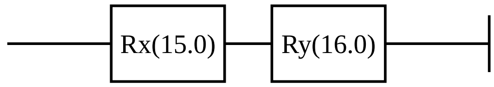
Presenting the kron block, where you can just input the gates on every qubit, one by one.




```
plot(chain(5, put(1:3 => kron(X,Y,Z)), put(3:5 => kron(H,Rx(15),T))))
```

The Rotation Gate

Its the general version of the Rx, Ry and Rz gates you saw above




```
plot(chain(1, rot(X, 15), rot(Y, 16)))
```