ALGORITMOS CUÁNTICOS EN QUIPPER

@mx_psi(@mstdn.io)

QUIPPER

INSTALACIÓN

`mx-psi.github.io/quantum-algorithms`

1. Instalar `stack`,
2. clonar `mx-psi/quantum-algorithms` y
3. ejecutar `make build`

EL LENGUAJE

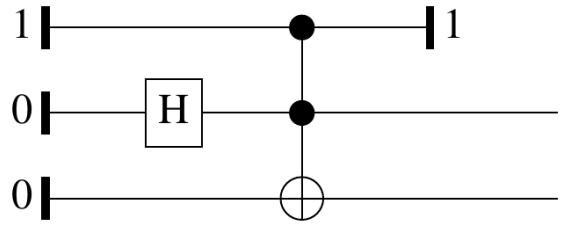
Quipper es un lenguaje embebido en Haskell que permite definir circuitos. Nos permite:

- Dar instrucciones para generar un circuito,
- generar los circuitos y
- ejecutarlos.

Todas las operaciones ocurren en una mónada `Circ`.

UN EJEMPLO DE CIRCUITO

```
bellPair :: Circ (Qubit, Qubit)
bellPair = do
  (x, y, z) <- qinit (True, False, False)
  y <- hadamard y
  (x,y,z) <- toffoli (x, y, z)
  qterm True x
  pure (y, z)
```



CIRCUITOS CLÁSICOS

EL ESPACIO DE ESTADOS

Un ordenador clásico hace cálculos con símbolos.

Ejemplo

Un **bit** es un elemento del espacio de estados

$$\mathbb{B} := \{0, 1\}.$$

En Quipper el tipo de los bits es `Bit`.

PUERTAS CLÁSICAS

Definición

Una **puerta clásica** es una función $f : \mathbb{B}^n \to \mathbb{B}^m$.

- n es el número de **entradas** y
- m es el número de **salidas**.

```
cgate_and :: [Bit] -> Circ Bit
cgate_not :: Bit -> Circ Bit
cinit :: Bool -> Circ Bit
cdiscard :: Bit -> Circ ()
```

CIRCUITOS

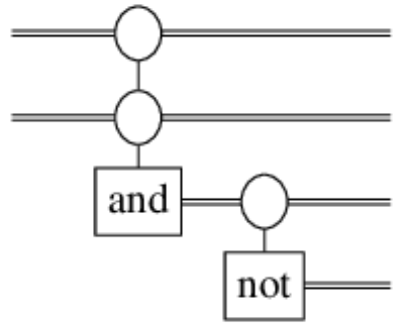
Definición

Un *circuito* es un grafo dirigido acíclico etiquetado con puertas, salidas y entradas, de tal forma que los grados coincidan.

Tiene una función $C : \mathbb{B}^n \to \mathbb{B}^m$ asociada.

EJEMPLO: NAND

```haskell
nand :: [Bit] -> Circ Bit
nand xs = do
  y <- cgate_and xs
  cgate_not y
```



FAMILIAS DE CIRCUITOS

Cuando necesitamos tener una entrada de tamaño arbitrario, consideramos una familia de circuitos $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$, tal que C_n tiene n entradas.

Su función asociada es $\mathcal{C}(x) = C_{|x|}(x)$.

Es *uniforme* si la función $n \mapsto C_n$ es computable.

CIRCUITOS REVERSIBLES

Para hacer un circuito reversible, llevamos $f : \mathbb{B}^n \to \mathbb{B}^m$ en

$$g(x, y) = (x, y \oplus f(x)).$$

En Quipper esto se hace con la función `classical_to_reversible`,

```
classical_to_reversible ::(qa -> Circ qb)
                        -> (qa, qb) -> Circ (qa, qb)
```

CIRCUITOS CUÁNTICOS

EL ESPACIO DE ESTADOS

El espacio de estados es un espacio de Hilbert separable complejo.

Ejemplo

Un **qubit** es un vector unitario de un espacio vectorial complejo con base ortonormal $\{|0\rangle, |1\rangle\}$,

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \qquad |\alpha|^2 + |\beta|^2 = 1$$

α y β son las **amplitudes** de $|\psi\rangle$.

Con n qubits, el espacio de estados es $Q^{\otimes n}$. En Quipper, el tipo de un qubit es `Qubit`.

¿QUÉ ES UNA PUERTA CUÁNTICA?

Una puerta cuántica es una aplicación unitaria

$$U : Q^{\otimes n} \to Q^{\otimes n}.$$

Ejemplo

La puerta de Hadamard se define

$$H|x\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^x|1\rangle)$$

En Quipper,

```
hadamard :: Qubit -> Circ Qubit
```

MEDICIÓN

La salida de un circuito cuántico será un vector unitario. Si medimos

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle$$

tenemos

$$P(\mathrm{Meas}\,|\psi\rangle = i) = |\alpha_i|^2$$

```
measure :: Qubit -> Circ Bit
```

MEDICIÓN Y ERROR

La salida de un circuito será un vector de amplitudes que muestreamos.

Definición

$$C : Q^{\otimes n} \to Q^{\otimes m} \text{ calcula } f : \mathbb{B}^n \to \mathbb{B}^m \text{ si}$$

$$P[C|x\rangle = f(x)] \geq \frac{2}{3}$$

GROVER

EL PROBLEMA

Dada una función $f : \mathbb{B}^n \to \mathbb{B}$ y el número

$$M = |\{x \ : \ f(x) = 1\}| > 0,$$

hallar una solución de

$$f(x) = 1.$$

La función se nos da en forma de un circuito (oráculo) que calcula

$$U|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle.$$

Lo encapsulamos en un tipo de datos `Oracle`.

OPERADOR DE DIFUSIÓN

El operador de difusión $D_n : Q^{\otimes n} \to Q^{\otimes n}$ consiste en

1. Aplicar la puerta de Hadamard a cada qubit,
2. aplicar un “cambio de fase” que lleva

$$|x\rangle \mapsto (-1)^{\delta_{x0}} |x\rangle$$

y

3. aplicar de nuevo la puerta de Hadamard a cada qubit.

```
diffusion :: [Qubit] -> Circ [Qubit]
diffusion = map_hadamard >=> phaseShift >=> map_hadamard
```

EL OPERADOR DE GROVER

Si $|\downarrow\rangle = H|1\rangle$ entonces

$$U|x\rangle|\downarrow\rangle = (-1)^{f(x)}|x\rangle|\downarrow\rangle$$

El operador de Grover consiste en

1. Aplicar el oráculo, fijando el último qubit a $|\downarrow\rangle$ y
2. aplicar el operador de difusión a todos salvo el resto de qubits.

```
groverOperator oracle (xs, y) = do
  (xs, y) <- circuit oracle (xs, y)
  xs      <- diffusion xs
  pure (xs, y)
```

INTERPRETACIÓN GEOMÉTRICA

El operador de Grover es la composición de dos reflexiones,

1. El oráculo refleja respecto a la suma uniforme de los vectores que no son soluciones y
2. el operador de difusión refleja respecto de la suma uniforme de las posibles entradas.

¡La composición de dos reflexiones es una rotación!

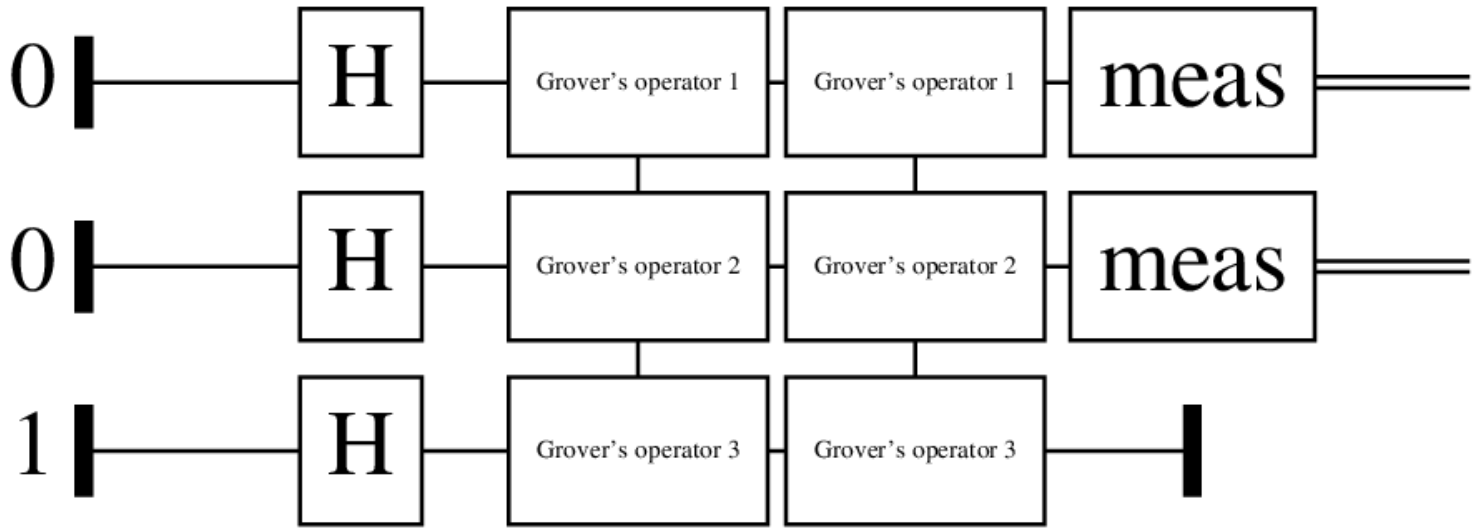
EL ALGORITMO DE GROVER

Rotamos la suma uniforme de todas las entradas hasta acercarnos a la suma uniforme de todas las entradas.

```
grover :: Int -> Oracle [Qubit] -> Circ [Bit]
grover m oracle = do
  (x, y) <- qinit (qc_false (shape oracle), True)
  (x, y) <- map_hadamard (x, y)
  (x, y) <- n `timesM` (groverOperator oracle) $ (x, y)
  qdiscard y
  measure x
  where n = ...
```

Se puede probar con el ejecutable `quantum`.

CIRCUITO


0
H
Grover's operator 1
Grover's operator 1
meas
0
H
Grover's operator 2
Grover's operator 2
meas
1
H
Grover's operator 3
Grover's operator 3

EXTRA

REFERENCIAS

- *Quantum Computation and Quantum Information* - Nielsen & Chuang
- *One Complexity Theorist's View of Quantum Computing* - Fortnow
- *Quantum Computational Complexity* - Watrous
- *An Introduction to Quantum Programming in Quipper* - Green et al.

QSHAPE

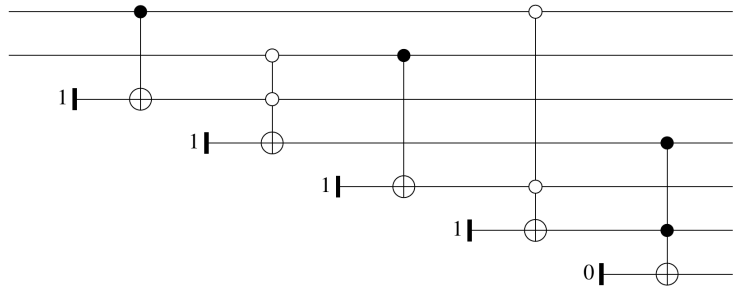
La clase de tipos `QShape` permite generalizar los tipos de las funciones:

```
instance QShape Bool Bit Qubit where
...

instance QShape ba ca qa => QShape [ba] [ca] [qa] where
...
```

GENERACIÓN DE CIRCUITOS

Podemos generar circuitos automáticamente a partir de funciones booleanas.

```
build_circuit
boolean_xnor (x,y) =
  (not x || y) &&  (x || not y)

xnor :: (Qubit,Qubit) -> Circ Qubit
xnor = unpack template_boolean_xnor
```



Classical.hs