

Alternative Entwicklungstools zu Qiskit - Cirq

Googles Quantum Computing Bibliothek für Python

Referent: Eric Brunk, 867437

Inhalt

- Was ist Cirq?
- Installation und Import
- NOT, CNOT und Hadamard
- Deutsch-Algorithmus
- Deutsch-Jozsa-Algorithmus

Was ist Cirq?

- Python Bibliothek von Google
- Aufbau und Optimierung von Quantenschaltkreisen
- Schaltkreise können auf Quantencomputern oder im Simulator ausgeführt werden

Quelle: [1]

Cirq ist eine Python Bibliothek von Google die es uns erlaubt Schaltkreise für den Quantencomputer zu erstellen, zu manipulieren und zu optimieren. Die Schaltkreise können auf einem Quantencomputer oder im Simulator ausgeführt werden.

Installation und Import

```
pip install cirq
```

```
import cirq
```

Die Installation erfolgt ganz einfach über pip.
Der Import gestaltet sich in Python auch sehr unkompliziert.

NOT



Quelle: [2]

Als ersten Schritt möchten wir versuchen einen Schaltkreis mit dem aus der Vorlesung bekannten NOT-Gatter und einer anschließenden Messung zu bauen.

NOT - Qiskit

```
# create Quantum Circuit with one Qubit  
circuit = qiskit.QuantumCircuit(1,1)  
  
# not gate and measure  
circuit.x(0)  
circuit.measure([0], [0])  
  
print("Circuit")  
circuit.draw()
```

16.01.2020

Eric Brunk - Quantum Computing

6

Da Cirq sehr ähnlich zu IBM's Qiskit ist, wird der Aufbau des Schaltkreises zuerst mit Qiskit gezeigt.

Es wird ein neuer QuantumCircuit erstellt. Der erste Parameter gibt die Anzahl der Qubits an. Der zweite Parameter die Anzahl der klassischen Bits, auf die die Messung später dargestellt wird.

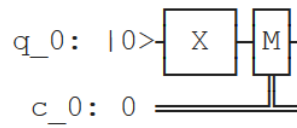
Für das NOT-Gatter genügt der Einsatz der Methode `x()`. Als Parameter wird der Index des gewünschten Qubits angegeben.

Für das Measure-Gatter wird die Methode `measure()` aufgerufen. Als ersten Parameter wird ein Array mit den Indizes der gewünschten Qubits angegeben, auf die das Gatter angewandt werden soll. Der zweite Parameter ist ein Array mit den Indizes der klassischen Bits, auf denen die Messung ausgegeben werden soll.

Im Anschluss wird über die Methode `draw()` unser Schaltkreis gezeichnet.

NOT - Qiskit

Circuit



Diese Folie stellt die Zeichnung des Schaltkreises in Qiskit dar. Verglichen wird mit der Darstellung die wir aus der VL kennen.

NOT - Cirq

```
# pick up a qubit
qubit = cirq.LineQubit(0) # position 0

# create a circuit
circuit = cirq.Circuit()
circuit.append(cirq.X(qubit)),
circuit.append(cirq.measure(qubit))

print("Circuit")
print(circuit)
```

16.01.2020

Eric Brunk - Quantum Computing

8

Der gleiche Schaltkreis wird nun mit Cirq gebaut.

In Cirq können die Qubits über `LineQubit()` einzeln erstellt werden. Der Parameter gibt die Position im Schaltkreis an.

Im Anschluss wird zuerst der eigentliche Schaltkreis erstellt.

Jetzt können wir mit der Methode `append()` unsere gewünschten Gatter hinzufügen.

Bei Cirq ist das NOT-Gatter ein großes `X()`. Als Parameter wird das gewünschte Qubit angegeben. Für unsere Messung das Gleiche.

Unseren Schaltkreis können wir jetzt über `print()` ausgeben.

NOT - Cirq

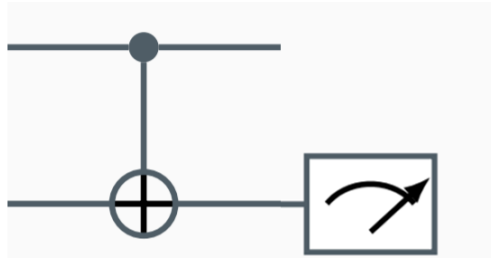
Circuit

0: —X—M—



Diese Folie stellt die Zeichnung des Schaltkreises in Cirq dar. Verglichen wird mit der Darstellung die wir aus der VL kennen.

CNOT



Quelle: [2]

Der nächste Schaltkreis den wir uns anschauen möchten ist CNOT mit anschließender Messung.

CNOT - Qiskit

```
# create Quantum Circuit with two Qubits  
circuit = qiskit.QuantumCircuit(2,1)
```

```
# cnot gate and measure  
circuit.cx(0, 1)  
circuit.measure([1], [0])
```

```
print("Circuit")  
circuit.draw()
```

Die Vorgehensweise in Qiskit ist dabei fast identisch zu der für den vorherigen Schaltkreis.

Allerdings benötigen wir nun zwei Qubits und nur ein klassisches Bit, da auch nur ein Qubit gemessen wird.

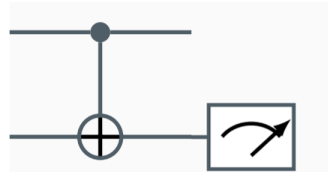
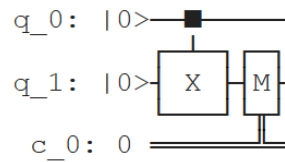
Das CNOT-Gatter sprechen wir über `cx()` an. Der erste Parameter gibt das sogenannte controll-bit an, der zweite das sogenannte target-bit.

Anschließend wird nur das target-bit gemessen.

Wir zeichnen wieder unseren Schaltkreis.

CNOT - Qiskit

Circuit



Diese Folie stellt die Zeichnung des Schaltkreises in Qiskit dar. Verglichen wird mit der Darstellung die wir aus der VL kennen.

CNOT - Cirq

```
# pick up qubits
qubit0 = cirq.LineQubit(0) # position 0
qubit1 = cirq.LineQubit(1) # position 1

# create a circuit
circuit = cirq.Circuit()
circuit.append(cirq.CNOT(control=qubit0, target=qubit1)),
circuit.append(cirq.measure(qubit1))

print("Circuit")
print(circuit)
```

16.01.2020

Eric Brunk - Quantum Computing

13

Den gleichen Schaltkreis möchten wir wieder in Cirq nachbauen.

Hierfür erstellen wir uns zwei Qubits.

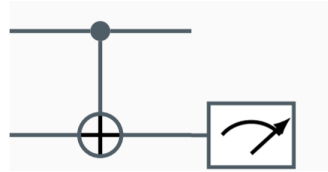
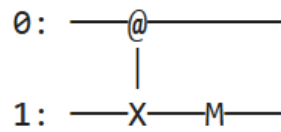
Auch hier ist der Vorgang fast identisch zum Beispiel mit dem NOT-Gatter.

Für Controlled Not wird jetzt die Methode CNOT() angerufen. Die Parameter sind hier dementsprechend benannt.

Auch diesen Schaltkreis geben wir wieder aus.

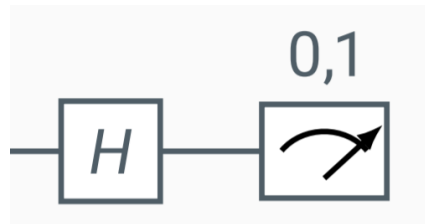
CNOT - Cirq

Circuit



Diese Folie stellt die Zeichnung des Schaltkreises in Cirq dar. Verglichen wird mit der Darstellung die wir aus der VL kennen.

Hadamard



Quelle: [2]

16.01.2020

Eric Brunk - Quantum Computing

15

Jetzt möchten wir uns einen Schaltkreis mit dem bekannten Hadamard-Gatter mit anschließender Messung anschauen.

Da mittlerweile klar sein sollte dass Qiskit sehr ähnlich zu Cirq ist, wird darauf verzichtet die Qiskit Beispiele auch zu implementieren.

Hadamard - Cirq

```
# pick a qubit
qubit = cirq.LineQubit(0) # position 0

# create a circuit
circuit = cirq.Circuit(
    cirq.H(qubit),
    cirq.measure(qubit)
)

print("Circuit")
print(circuit)
```

16.01.2020

Eric Brunk - Quantum Computing

16

Wir erzeugen wieder ein Qubit und unseren Circuit.

Hier sehen wir, dass wir die gewünschten Gatter auch direkt in die Klammern des Circuits schreiben können um sie hinzuzufügen.

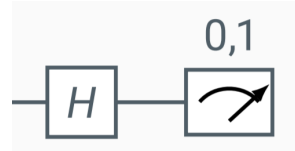
Wir fügen Hadamard mit `H()` und eine anschließende Messung hinzu.

Danach geben wir den Schaltkreis wieder aus.

Hadamard - Cirq

Circuit

0: —H—M—



Diese Folie stellt die Zeichnung des Schaltkreises in Cirq dar. Verglichen wird mit der Darstellung die wir aus der VL kennen.

Hadamard - Cirq

```
# simulate
simulator = cirq.Simulator()
result = simulator.run(circuit, repetitions=1000)

print("Result")
print(result)
```

```
Result
0=111001011101101010111110001111111000110101110011101001110001101001100000001011110000100010100101000000
01001011111011111101010111101000110011011001010010011011110010111100101011010000000100000010100010101010
001110001011000100110111001001011100001101111100000110010010111110011110101110111000101000001101010100
10011110101000011011000101001101000110111110001111011100010010110010100001011011010000011110100111110
01110010101111110011100100111100011111100000001100000000011001000110110010010111000000111000011011110
011000010111100100010011011011011001111101010110101000111001000010110110101000010000110101100001011
10111110110011011000000111010001110011101110001010010100010000001011001111101100000000111100110
010101010111110000011000110110110110010000000111011001000010000001101011001011110100100001010110011
1011111101011001100101010010110101001110011110100000100011011100001011001100001001111011000011010100110100
10110011011110010001101001000101101101
```

Da das Hadamard-Gatter den Zustand unseres Qubits in eine Superposition überführt interessiert uns natürlich unsere Messung.

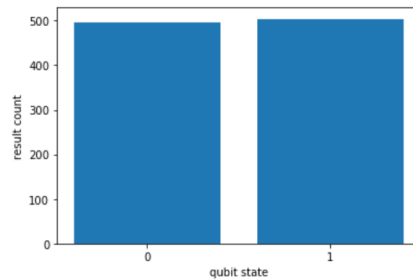
Unseren Schaltkreis können wir in einem Simulator laufen lassen.

Als Anzahl an Wiederholungen wurde hier 1000 gewählt.

Unser Ergebnis zeigen 1000 Werte mit einer Verteilung von 0 und 1.

Hadamard - Cirq

```
# histogram  
cirq.plot_state_histogram(result)
```



```
array([496., 504.])
```

16.01.2020

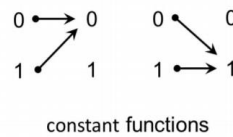
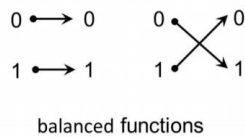
Eric Brunk - Quantum Computing

19

Um unter Ergebnis auch plotten zu können verwenden wir die von cirq mitgelieferte Methode `plot_state_histogram()`.

Wir sehen in unserem Diagramm, dass wir eine ungefähre Verteilung von 50/50 haben.

Deutsch Algorithmus

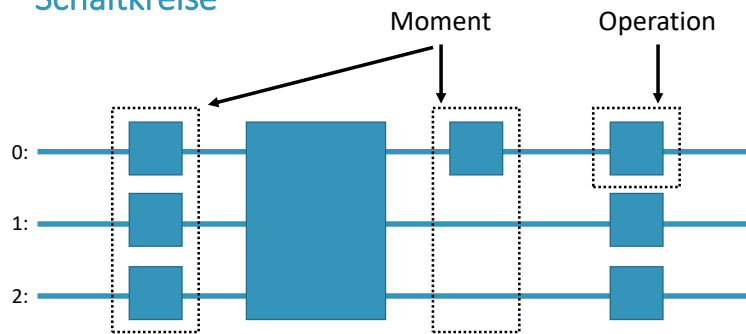


Quelle: [2]

Der Deutsch-Algorithmus kann mit nur einem Durchlauf erkennen, ob eine Funktion balanciert oder konstant ist.

Klassische Rechner benötigen dafür mindestens zwei oder mehr Durchläufe.

Schaltkreise



16.01.2020

Eric Brunk - Quantum Computing

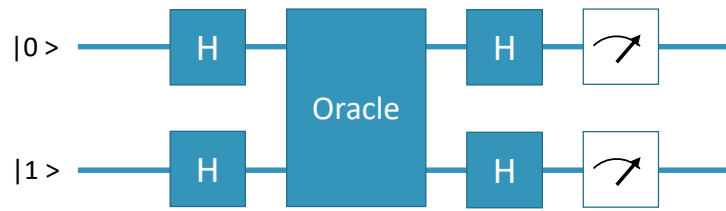
21

Ein Schaltkreis in Cirq kann in sogenannte Moments und Operations unterteilt werden.

Ein Moment spezifiziert eine Sammlung von Gattern zu einem gewissen Moment.

Operations sind die eigentlichen Gatter.

Deutsch Algorithmus



16.01.2020

Eric Brunk - Quantum Computing

22

Diese Folie zeigt, wie der Schaltkreis für den Deutsch-Algorithmus aufgebaut ist.

Wir haben zwei Qubits, eines im Zustand 0 und eines im Zustand 1.

Beide Qubits werden durch ein Hadamard-Gatter in die Superposition überführt.

Danach folgt die eigentliche Funktion, Oracle-Funktion genannt, die entweder balanciert oder eben konstant ist.

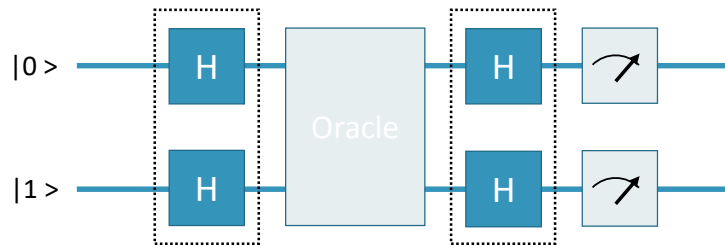
Anschließend werden beide Qubits wieder jeweils durch ein Hadamard-Gatter geschickt.

Am Ende werden beide Qubits gemessen.

Das zweite Hadamard-Gatter und die Messung sind beim zweiten Qubit eigentlich überflüssig.

Der Vollständigkeit halber wurden sie aber hinzugefügt.

Deutsch Algorithmus



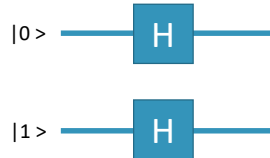
16.01.2020

Eric Brunk - Quantum Computing

23

Der Schaltkreis kann unterteilt werden in Hadamard-Moment, Oracle-Funktion und Mess-Moment.

Deutsch Algorithmus



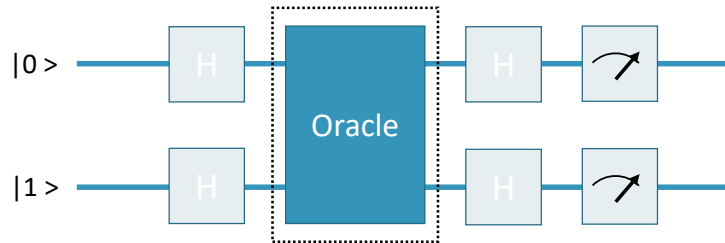
```
# create hadamard moment  
moment_h_all = circ.Moment( [circ.H(qubit0), circ.H(qubit1)] )
```

Diese Folie zeigt die Implementierung des Hadamard-Moments.

Es wird ein neuer Moment erstellt, dem ein Array an Hadamard-Gattern zugewiesen wird.

Dieser Moment kann beliebig oft wiederverwendet werden.

Deutsch Algorithmus



Deutsch Algorithmus – Oracle Gatter

```
# oracle gate
def get_oracle_gate(oracle):
    # if oracle is constant
    if oracle == "c":
        c = np.random.randint(2)
        if c == 1:
            return [cirq.I(qubit0), cirq.X(qubit1)]
        else:
            return [cirq.I(qubit0), cirq.I(qubit1)]
    # if oracle is balanced
    else:
        return cirq.CNOT(control=qubit0, target=qubit1)
```

16.01.2020

Eric Brunk - Quantum Computing

26

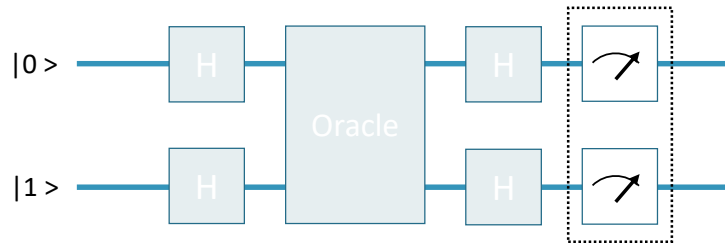
Diese Folie zeigt die Implementierung der Oracle-Funktion für den Deutsch-Algorithmus.

Es wird die Methode `get_oracle_gate(oracle)` erstellt die einen Parameter („c“ für konstant oder „b“ für balanced) entgegen nimmt.

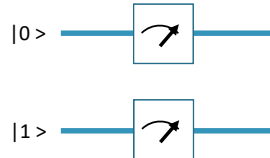
Wenn ein „c“ übergeben wird, gibt die Methode abhängig von einer generierten Zufallszahl entweder das Gatter [Identity, NOT] oder [Identity, Identity] zurück.

Wird ein „b“ übergeben gibt die Methode eine Möglichkeit einer balancierten Funktion in Form von CNOT zurück.

Deutsch Algorithmus



Deutsch Algorithmus



```
# create measure moment  
moment_m_all = cirq.Moment( [cirq.measure(qubit0), cirq.measure(qubit1)] )
```

Diese Folie zeigt die Implementierung des Mess-Moments.

Es wird ein neuer Moment erstellt, dem ein Array an Messungen zugewiesen wird.

Auch dieser Moment kann beliebig oft wiederverwendet werden.

Deutsch Algorithmus – Schaltkreis

```
# circuit
circuit = cirq.Circuit()
circuit.append(cirq.X(qubit1))
circuit.append(moment_h_all)
# insert oracle gate
circuit.append(get_oracle_gate("b"))

circuit.append(moment_h_all)
circuit.append(moment_m_all)

print("Result")
print(result)
```

16.01.2020

Eric Brunk - Quantum Computing

29

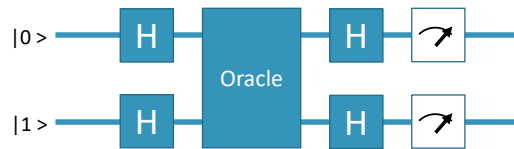
Diese Folie zeigt, wie die einzelnen Moments und die Oracle-Funktion zu einem Schaltkreis zusammengebaut werden.

Zuerst muss das letzte Qubit mit dem NOT-Gatter in den Zustand 1 gebracht werden.

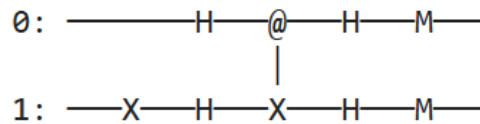
Danach folgen die eigentlichen Moments bzw. die Oracle-Funktion.

Am Ende geben wir unseren Schaltkreis wieder aus.

Deutsch Algorithmus – Schaltkreis



Circuit



16.01.2020

Eric Brunk - Quantum Computing

30

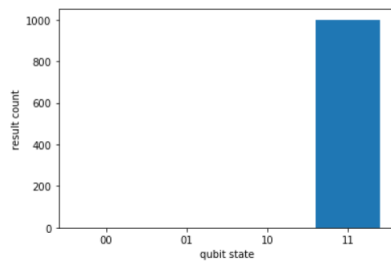
Diese Folie zeigt zum Vergleich:

Oben -> Schaltkreis den wir erreichen wollten

Unten -> Schaltkreis den uns Cirq ausgibt.

Deutsch Algorithmus – Histogram

```
# simulate and plot  
simulator = cirq.Simulator()  
result = simulator.run(circuit, repetitions=1000)  
cirq.plot_state_histogram(result)
```



```
array([ 0.,  0.,  0., 1000.])
```

16.01.2020

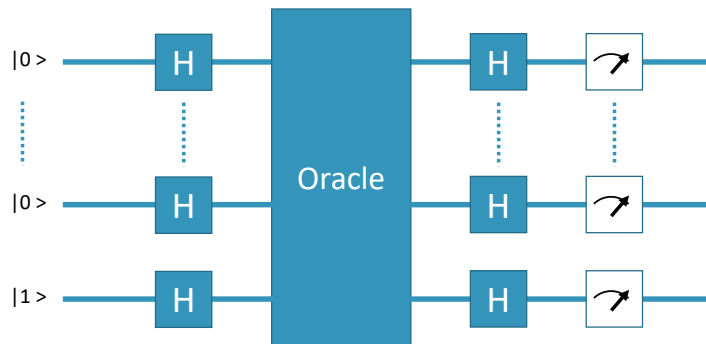
Eric Brunk - Quantum Computing

31

Unseren Schaltkreis wollen wir natürlich auch simulieren.

Da wir eine balancierte Funktion in den Schaltkreis eingefügt haben, gibt uns der Schaltkreis eine 100% Verteilung von 11 zurück.

Deutsch-Jozsa Algorithmus



16.01.2020

Eric Brunk - Quantum Computing

32

Der Deutsch-Jozsa-Algorithmus erweitert den Deutsch-Algorithmus dadurch dass wir das gleiche Vorgehen nun mit n -Qubits erreichen wollen.

Diese Folie zeigt ungefähr, wie solch ein Schaltkreis aussehen sollte.

Auch hier kann das Hadamard-Gatter und die Messung des letzten Qubits weggelassen werden.

Deutsch-Jozsa Algorithmus – Gatter

```
# n qubits
n_qubits = 5
# pick up qubits
qubits = [cirq.LineQubit(x) for x in range(n_qubits+1)]

# gates
gate_x_last = cirq.X(qubits[n_qubits])
gate_h_all = [cirq.H(qubit) for qubit in qubits]
gate_m_all = [cirq.measure(qubit) for qubit in qubits]
gate_h_all_ex_last = [cirq.H(qubit) for qubit in qubits[:-1]]
gate_m_all_ex_last = [cirq.measure(qubit) for qubit in qubits[:-1]]
```

16.01.2020

Eric Brunk - Quantum Computing

33

Die vorherige Implementierung wird dadurch erweitert, dass die Anzahl an Qubits über ein Array, abhängig der Variable `n_qubits` erfasst wird.

Die eigentlichen Gatter können dann auf die gewünschten Einträge des Arrays angewandt und in eine Variable gespeichert werden.

Deutsch-Jozsa Algorithmus – Oracle Gatter

```
# oracle gate
def get_oracle_gate(oracle):
    rnd = np.random.randint(2)
    # if oracle is constant
    if oracle == "c":
        if rnd == 1:
            gate = [cirq.I(qubit) for qubit in qubits[:-1]]
            gate.append(cirq.X(qubits[n_qubits]))
            return gate
        else:
            return [cirq.I(qubit) for qubit in qubits]
    # if oracle is balanced
    else:
        if rnd == 1:
            gate = [cirq.CNOT(control=qubit, target=qubits[n_qubits]) for qubit in qubits[:-1]]
            gate.append(cirq.X(qubits[n_qubits]))
            return gate
        else:
            return [cirq.CNOT(control=qubit, target=qubits[n_qubits]) for qubit in qubits[:-1]]
```

16.01.2020

Eric Brunk - Quantum Computing

34

Diese Folie zeigt die Implementierung der Oracle-Funktion für den Deutsch-Jozsa-Algorithmus.

Der hauptsächliche Unterschied hierbei besteht einfach darin, dass die Gatter für die konstante und die balancierte Funktionen eben auf das Array von Qubits und nicht wie vorher auf zwei spezifische Qubits angewandt werden.

Bei den balancierten Funktionen sind in diesem Beispiel nur zwei Möglichkeiten implementiert.

Deutsch-Jozsa Algorithmus – Schaltkreis

```
# circuit
circuit = cirq.Circuit()
circuit.append(moment_x_last)
circuit.append(moment_h_all)

circuit.append(get_oracle_gate("b"))

circuit.append(moment_h_all_ex_last)
circuit.append(moment_m_all_ex_last)

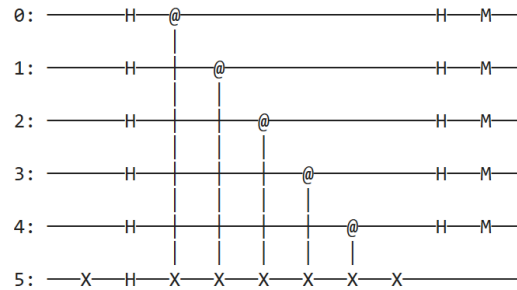
print("Result")
print(result)
```

Auch hier können die verschiedenen Moments einfach im Circuit zusammengeführt werden.

Danach erfolgt wieder die Ausgabe des Schaltkreises.

Deutsch-Jozsa Algorithmus – Schaltkreis

Circuit



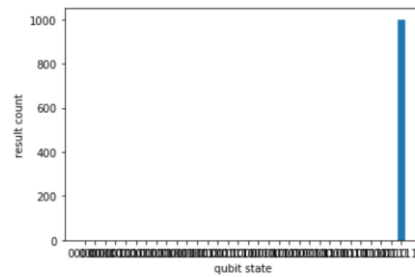
16.01.2020

Eric Brunk - Quantum Computing

36

Diese Folie zeigt die Ausgabe des Schaltkreises für die DJ-Algorithmus am Beispiel von einer balancierten Funktion in Form von CNOT.

Deutsch Algorithmus – Histogram



```
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  
        0.,  0.,  0.,  0., 1000.] )
```

Das dazu gehörige Histogramm zeigt auch hier eine 100%-Verteilung des erwarteten Wertes bei 5 Qubits.

Git-Repository

Die hier gezeigte Implementierung und die gesamte Präsentation befindet sich in einem Git-Repository unter dem folgenden Link:

https://github.com/erxn91/quantum_computing

Quellen

Literatur:

[1] – Cirq Documentation, url: <https://cirq.readthedocs.io/en/stable/>

Bilder:

[2] – Prof. Dr. Jörg Hettel, Quantum Computing and Information An Introduction for Computer Scientists – Quantum Algorithms

Alle Informationen über Cirq stammen aus der Dokumentation der Bibliothek.

Die mit [2] gekennzeichneten Bilder stammen aus dem Skript von Prof. Dr. Jörg Hettel – Quantum Computing and Information, Kapitel Quantum Algorithms.

Fragen?