BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
LEIPZIG
UNIVERSITY OF COOPERATIVE EDUCATION

# Quantencomputing
—
# Meyer-Penny-Spiel

**Hausarbeit**

im Studiengang Informatik

an der Berufsakademie Sachsen – Staatliche Studienakademie Leipzig

im Modul

**Algorithmen und Datenstrukturen**

von

**Florian Fröhlich**

und

**Christoph Rosenau**

1. Juni 2023

**Matrikelnummern:** 5002155 – Florian Fröhlich
5002078 – Christoph Rosenau

**Gutachter:** Prof. Dr. Holger Perlt

# Inhaltsverzeichnis

# 1 Quantencomputing als Konzept

Quantencomputing ist, grundsätzlich gesehen, Informationsverarbeitung mittels quantenmechanischer Systeme [3, S. 1]. Diese Technologie ermöglicht es, gewisse Berechnungen vielfach schneller und effizienter als auf herkömmlichen, *klassischen* Computern vorzunehmen. Dabei wird ein grundlegendes Konzept der Quantenmechanik nutzbar gemacht, die *Superposition* – der Umstand, dass in einem Quantensystem ein Zustand erst im Rahmen der Beobachtung determiniert wird, davor jedoch verschiedene Zustände sich zugleich überlagern können.

## 1.1 Unterschied Bit – Qbit

Dieses Konzept der *Superposition* wird im Quantencomputing dergestalt angewandt, dass die grundlegende atomare Einheit des Quantencomputings nicht, wie im Digitalrechner, das Bit ist, sondern das sogenannte *QBit* (Quantenbit). Auch dies ist ein binäres System, jedoch jeweils erst ab der Messung. Vor der Messung kann ein Qbit eine unendliche Zahl an Zuständen zwischen den möglichen Messzuständen 0 und 1 einnehmen. Mathematisch werden diese „Möglichkeiten“ durch Vorfaktoren (üblicherweise $\alpha$ und $\beta$) ausgedrückt:

$$\alpha \cdot |0\rangle + \beta \cdot |1\rangle \tag{1.1}$$

wobei

$$|\alpha|^2 + |\beta|^2 = 1 \qquad \{\alpha, \beta\} \in \mathbb{C} \tag{1.2}$$

$\alpha$ und $\beta$ werden hierbei als *Amplituden* des QBits (1.1) bezeichnet. Die Amplituden bilden gemeinsam mit der Superposition $|0\rangle + |1\rangle$ einen Vektorraum, der Zustandsvektor $(\alpha, \beta)$

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{1.3}$$

ist die Linearkombination der zweidimensionalen Standardbasisvektoren[1, S. 22]:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \alpha \cdot |0\rangle + \beta \cdot |1\rangle \tag{1.4}$$

# 2 Aufgabe 3

## 2.1 Grundlagen

Als Beispiel sei ein QBit mit diesen Amplituden angenommen:

$$q = \sqrt{\frac{1}{3}}\,|0\rangle - \sqrt{\frac{2}{3}}\,|1\rangle \tag{2.1}$$

Die Wahrscheinlichkeiten, nach einer Messung 0 oder 1 zu erhalten, ergeben sich aus:

$$\mathcal{P}(q_0) = \left(\left|\sqrt{\frac{1}{3}}\right|\right)^2 = \frac{1}{3} \tag{2.2}$$

$$\mathcal{P}(q_1) = \left(\left|-\sqrt{\frac{2}{3}}\right|\right)^2 = \frac{2}{3} \tag{2.3}$$

## 2.2 Transformationsmatrix

Im Programmcode zu dieser Arbeit ist dieses QBit mittels der Methode `Aufgaben.aufgabe1()` und der dort verwendeten `Qbit`-Klasse verwirklicht. Das gesamte Programm ist in `Java` geschrieben. Bewusst wurde auf die Verwendung eines Quantencomputing-Frameworks wie `qiskit` verzichtet, um die tatsächlichen mathematischen Gatteroperationen nachzuvollziehen. Die Amplituden $\alpha = \sqrt{\frac{1}{3}}$ und $\beta = -\sqrt{\frac{2}{3}}$ wurden nicht vorgegeben *(hardgecodet)*, sondern zu Übungszwecken selbst durch Matrixmanipulation erzeugt. (Nachteilig an diesem Ansatz ist, dass bei keiner der implementierten Quantenoperationen ein tatsächliches Testen an echten Quantencomputern möglich ist, bei dem man auch die erwartbare Fehlerquote der Operationen hätte beobachten können.)

Der Initialzustand eines neuen QBits aus der `Qbit`-Klasse entspricht $1\,|0\rangle + 0\,|1\rangle$, dies wird im Konstruktor `Qbit.Qbit()` festgelegt. Ist das Ziel, eine Matrix zu finden, die die gewünschten $\alpha$- und $\beta$-Werte erzeugen, dann gilt:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \sqrt{\frac{1}{3}} \\ -\sqrt{\frac{2}{3}} \end{pmatrix} \tag{2.4}$$

mit

$$a = \sqrt{\frac{1}{3}} \tag{2.5}$$

$$c = -\sqrt{\frac{2}{3}} \tag{2.6}$$

Da die gesuchte Matrix unitär sein muss [4, S. 265] gilt:

$$a^2 + b^2 = 1 \tag{2.7}$$

$$c^2 + c^2 = 1 \tag{2.8}$$

$$ac + bd = 0 \tag{2.9}$$

Durch Einsetzen ergeben sich jeweils 2 Lösungen:

$$\left(\sqrt{\frac{1}{3}}\right)^2 + b^2 = 1 \tag{2.10}$$

$$b = \pm\sqrt{\frac{2}{3}} \tag{2.11}$$

$$\left(-\sqrt{\frac{2}{3}}\right)^2 + d^2 = 1 \tag{2.12}$$

$$d = \pm\sqrt{\frac{1}{3}} \tag{2.13}$$

Wähle $b$ und $d$:

$$b = -\sqrt{\frac{2}{3}} \tag{2.14}$$

$$d = -\sqrt{\frac{1}{3}} \tag{2.15}$$

$$\tag{2.16}$$

Nach Einsetzen in 2.4 ergibt via 2.9 die gesuchte Matrix:

$$\begin{pmatrix} \sqrt{\frac{1}{3}} & -\sqrt{\frac{2}{3}} \\ -\sqrt{\frac{2}{3}} & -\sqrt{\frac{2}{3}} \end{pmatrix} \tag{2.17}$$

Diese ist tatsächlich unitär:

$$\begin{pmatrix} \sqrt{\frac{1}{3}} & -\sqrt{\frac{2}{3}} \\ -\sqrt{\frac{2}{3}} & -\sqrt{\frac{2}{3}} \end{pmatrix} \begin{pmatrix} \sqrt{\frac{1}{3}} & -\sqrt{\frac{2}{3}} \\ -\sqrt{\frac{2}{3}} & -\sqrt{\frac{2}{3}} \end{pmatrix} = \begin{pmatrix} \sqrt{\frac{1}{3}} + \sqrt{\frac{2}{3}} & -\sqrt{\frac{2}{9}} + \sqrt{\frac{2}{9}} \\ -\sqrt{\frac{2}{9}} - \sqrt{\frac{2}{9}} & \sqrt{\frac{2}{3}} + \sqrt{\frac{1}{3}} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.18}$$

Die Matrix kann als nichtorthodoxes Quantengatter verstanden werden, welches das gewünschte $\alpha$ und $\beta$ aus dem Initialzustand erzeugt. Dieser Vorgang ist im Code via `Qbit.thirdgate()` umgesetzt.

## 2.3 Messlogik

Tatsächlich ergibt eine vielmalige Messung mittels `Qbit.measure()` eines so simulierten QBits eine ungefähre Aufteilung in $\frac{1}{3}$ mal 0 und $\frac{2}{3}$ mal 1.

`Qbit.measure()` ist dergestalt implementiert, dass eine Pseudozufallsvariable im Intervall $[0, 1]$ erzeugt wird und durch eine `if-else`-Verzweigung mit dem Quadrat von $\alpha$ verglichen wird. Ist die Zufallszahl kleiner als $\alpha^2$, wird 0 zurückgegeben, dies entspricht dem Messwert des QBits. Andernfalls kann durch $|\alpha|^2 + |\beta|^2 = 1$ vom Messwert 1 ausgegangen werden (`else`-Zweig).

# 3 Das Meyer-Penny-Spiel

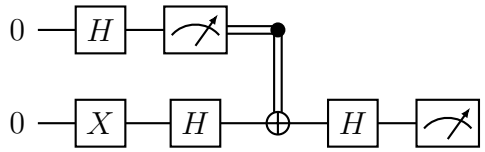
## 3.1 Grundlage

Das Meyer-Penny-Spiel wurde 1999 als Gedankenexperiment von David A. Meyer vorgestellt, um Auswirkungen gewisser Quantencomputing-Algorithmen auf Bereiche der Spieltheorie zu verdeutlichen [2]. In vereinfachter Form beschreibt es das geminsame Werfen einer Münze durch zwei Spieler, die jeweils auf ihre Münzseite hoffen. Nur einer der beiden Spieler (bei Meyer: $Q$) kann auf die Mittel des Quantencomputings zugreifen, der andere muss sich auf sein Zufallsglück verlassen (bei Meyer: *Picard,* beide aus dem *StarTrek*-Universum).

Das Grundprinzip ist einfach beschrieben: „P[icard] is to place a penny, head up, in a box, whereupon they will take turns (Q, then P, then Q) flipping the penny over (or not), without being able to see it. Q wins if the penny is head up when they open the box [2, S. 1052].“ Unabhängig von seiner gewählten Spielstrategie (flippen oder nicht) könnte Picard hier auf eine akzeptable Gewinnwahrscheinlichkeit von $\mathcal{P}(P_1) = \frac{1}{2}$ hoffen, während die Gewinnwahrscheinlichkeit Qs bei jeder seiner vier möglichen Strategien bei $\mathcal{P}(Q_1) = 2 \cdot \frac{1}{4}$ liegen müsste.

## 3.2 Quantenschaltung

Die Besonderheit liegt aber darin, dass Q bei seinen beiden Zügen die Münze weder flippt noch nichtflippt, sondern in den Zustand der Superposition versetzt. Das entsprechende Quantengatter ist das *Hadamard-Gate,* der Spielablauf lässt sich in Gattern folgendermaßen ausdrücken:

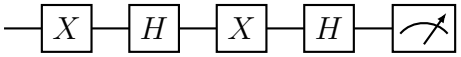



Das obere QBit repräsentiert schlicht Picards Entscheidung, die hier der Einfachheit ebenfalls als zufällig angenommen wird und durch ein Hadamard-Gate repräsentiert wird, dass sofort ausgelesen wird und den Zustand 0 oder 1 an das zweite QBit übergibt. Diesem zweiten QBit entspricht die eigentliche Münzflipp-Aktion. Das X-Gate setzt das Qbit

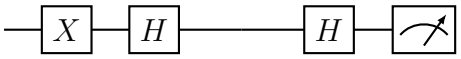
auf 1, bevor es im ersten Hadamard-Gate in Superposition gebracht wird. Die dann folgende Eingabe von Picards Entscheidung via CNOT ist für das Endergebnis tatsächlich irrelevant, denn nach dem anschließenden Hadamard-Gate wird das QBit immer jenen Zustand messen lassen, den es zuvor beim X-Gate hatte.

Die Schaltung lässt sich vereinfachen, wenn man das erste QBit durch eine Fallunterscheidung ersetzt. Wenn Picard sich entscheidet, die Münze zu flippen, dann wird statt des CNOT-Gates ein X-Gate angewandt, ansonsten wird es ausgelassen:

Fall 1: Picard flippt:




Fall 2: Picard flippt nicht:



## 3.3 Mathematische Operationen

In dieser Schaltung werden nur noch zwei verschiedene Gatter verwendet:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3.1}$$

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{3.2}$$

Die Matrixmultiplikation im Fall 1 ist damit (Multiplikation von rechts nach links, ganz rechts der Eingangsvektor des QBits):

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{3.3}$$

$$= \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{3.4}$$

$$= \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{3.5}$$

$$= \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{3.6}$$

$$= \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} \tag{3.7}$$

$$= \frac{1}{2} \begin{pmatrix} -1+1 \\ -1-1 \end{pmatrix} \tag{3.8}$$

$$= \frac{1}{2} \begin{pmatrix} 0 \\ -2 \end{pmatrix} \tag{3.9}$$

$$= \begin{pmatrix} 0 \\ -1 \end{pmatrix} \tag{3.10}$$

$$\text{gemessen als} \Longrightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{3.11}$$

Verzichtet Picard auf das Flippen, ist die Rechnung kürzer:

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} \tag{3.12}$$

$$= \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} \tag{3.13}$$

$$= \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0\\ 1\end{pmatrix} \tag{3.14}$$

$$= \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ -1\end{pmatrix} \tag{3.15}$$

$$= \frac{1}{2}\begin{pmatrix}0\\ 2\end{pmatrix} \tag{3.16}$$

$$= \begin{pmatrix}0\\ 1\end{pmatrix} \tag{3.17}$$

Es zeigt sich, dass, unabhängig von Picards Entscheidung, in jedem der beiden Fälle immer das gleiche Ergebnis folgt. Angewandt auf das Gedankenexperiment bedeutet das, dass immer der Kopf nach dem Spiel nach oben zeigen wird und Q immer gewinnen wird.

## 3.4 Implementierung im Programmcode

Wie oben bereits erwähnt, wird ein QBit durch Implementierung der `Qbit`-Klasse im Zustand $|0\rangle$ instanziiert. Dort sind auch die verwendbaren Gatter als Methoden angelegt. Diese sind der Einfachheit halber nur für reelle Zahlenoperationen verwendbar. `Qbit.flip()` entspricht funktional dem X-Gatter, `Qbit.hadamardGate()` ist selbsterklärend.

Die Logik des Meyer-Penny-Spiels wird dann in der `Aufgaben`-Klasse durchgeführt (`meyerPennySpiel()`). Instanziiert werden hier zwei QBits (`picard` und `penny`), wobei `picard` der zufälligen Entscheidung Picards entspricht, ob er die Münze flippt oder nicht, `penny` entspricht der oben berechneten Spiellogik. Das `penny`-Qbit wird zunächst geflippt, um es in die Position entsprechend $|1\rangle$ zu bringen. Dann wird es mit `hadamardGate()` in Superposition gebracht. Die Logik des CNOT-Gates, das nun folgen würde, wurde auch hier mit einer Fallunterscheidung umgesetzt: Wenn das aus der Superposition herausgemessene `picard`-QBit einer 1 entspricht, dann wird auf `penny` eine weitere `flip()`-Operation angewandt. Danach folgt nochmals `penny.hadamardGate()` und die Münze kann ausgemessen werden. Eine 1 wird als *Kopf* interpretiert, eine 0 als *Zahl*. Mittels der Zähllogik kann

gezeigt werden, dass bei jeder Iteration der `meyerPennySpiel()`-Methode das Ergebnis *Kopf* erzielt wird.

Quantencomputer gelten als Rechner, die im Gegensatz zu klassischen Computern echte Zufallszahlen erzeugen können. Das Meyer-Penny-Spiel zeigt jedoch, dass nicht immer, wenn Quantencomputer im Spiel sind und eine Zufallsentscheidung vorgeblich getroffen wird, diese auch tatsächlich zufällig sein muss.

# Literatur


[1] Matthias Homeister. *Quantum Computing verstehen: Grundlagen – Anwendungen – Perspektiven*. Wiesbaden, 2018.

[2] David A. Meyer. „Quantum Strategies“. In: *Phys. Rev. Lett.* 82.5 (1999), S. 1052–1055. DOI: 10.1103/PhysRevLett.82.1052.

[3] Michael A. Nielsen und Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge, 2011.

[4] Robert S. Sutor. *Dancing with Qubits: How quantum computing works and how it can change the world*. Birmingham, 2019.

# Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder veröffentlicht, noch einer anderen Prüfungsbehörde vorgelegt.

Leipzig, 1. Juni 2023