Fakultät für Mathematik,
Physik und Informatik
Mathematisches Institut

Prinzipien des Quantencomputers und der Algorithmus von Shor

Diplomarbeit
von
Wolfgang Riedl

Neusorg, den 29. März 2011

Aufgabenstellung:
Prof. Dr. Frank Lempio

Einleitung

Heutige Rechner sind zwar wesentlich stärker, basieren aber immer noch auf den gleichen Prinzipien wie die zu Beginn des 19. Jahrhunderts eingeführten Rechenmaschinen.
1982 veröffentlichte Richard Feynman Überlegungen darüber, dass quantenmechanische Systeme effizient simuliert werden können, indem man einen „*Quantensimulator*" nutzt, welcher selbst auf einem quantenmechanischem System aufbaut. David Deutsch, der als einer der Väter des Quantencomputers gilt, führte diese Idee weiter und modellierte 1985 eine *Quantenturingmaschine* - die Idee zum Quantencomputer war geboren (vgl. [13] Kapitel 10.6).
In den darauf folgenden Jahren wurden einige interessante Algorithmen für Quantencomputer vorgestellt, wie z.B. ein Zufallsgenerator, der echte Zufallszahlen erzeugen kann, es wurde jedoch angezweifelt, ob Quantencomputer jemals einen praktischen Nutzen liefern würden.
Dies änderte sich schlagartig, als Peter Shor 1994 einen Algorithmus vorstellte, mit dessen Hilfe man natürliche Zahlen durch den Einsatz eines Quantencomputers effizient faktorisieren kann. Da z.B. die Sicherheit des RSA-Verschlüsselungsverfahrens darauf beruht, dass die Faktorisierung großer Zahlen nur mit großem Aufwand durchführbar ist, hat der Algorithmus von Shor den Quantencomputer als Forschungsgebiet wieder attraktiver werden lassen. In der gleichen Arbeit zeigt Shor auch noch eine Möglichkeit durch den Einsatz eines Quantencomputers den diskreten Logarithmus zu berechnen, was bis zu diesem Zeitpunkt ebenfalls als „kompliziert" galt und ebenfalls die Sicherheit eines beliebten Kryptosystems (Elgamal) gefährdet. Weiter kann man zeigen, dass Quantenalgorithmen nicht nur zur Faktorisierung oder Berechnung des diskreten Logarithmus verwendet werden können, sondern auch bei anderen Problemen, wie z.B. bei der Suche in Datenbanken Vorteile bietet. Ein sehr prominentes Beispiel hierfür ist der von Lov K. Grover vorgestellte Algorithmus zur effizienten Suche in einer nicht sortierten Datenbank (siehe [12]). Die Grundidee hinter diesem Algorithmus ist die sogenannte *Amplitudenverstärkung* (englisch: *amplitude amplification*):
Man startet mit einem Zustand in dem bei einer Messung alle Datenbankeinträge mit gleicher Wahrscheinlichkeit gemessen werden. Danach wird mittels eines iterativen Verfahrens der Zustand so geändert, dass die Wahrschein-

lichkeit der ,,richtigen" Lösung sukzessive vergrößert wird und die Wahrscheinlichkeit der ,,falschen" Lösungen gegen 0 geht. Nach einer gewissen Anzahl an Iterationen wurde der Zustand so modifiziert, dass nur noch das richtige Ergebnis gemessen werden kann.
Ein weiteres interessantes Einsatzgebiet für Quantencomputer könnte auch in der Computergrafik zu finden sein. Einige bekannte klassische Verfahren können auf ein Suchproblem zurückgeführt werden, welches mit Varianten von Grovers Algorithmus gelöst werden kann (vgl. [16] Kapitel 4.2):

- *Z-Buffering*: Die grundlegende Idee hinter diesem Renderingverfahren ist, dass für jeden zu zeichnenden Pixel über die zu zeichnenden Objekte iteriert wird und der Abstand zu diesem Pixel berechnet wird. Findet man in diesem Schritt heraus, dass das Objekt näher am Schirm liegt als alle bisher Gefundenen wird die Farbe des Objektes für den aktuellen Pixel verwendet. Mittels eines Quantencomputers könnte das näheste Objekt ermittelt werden, womit man sich die Iteration über alle Objekte sparen kann.

- *Raytracing*: Ein weiteres sehr spannendes Verfahren zur Erzeugung von Computergrafiken ist das sogenannte Raytracing. Die Idee hinter diesem Verfahren ist, dass man ,,Sehstrahlen" von einem virtuellen Kamerapunkt aus in Richtung des Schirms schickt und berechnet welches Objekt in welcher Entfernung von diesem Strahl getroffen wird und hiermit die Farbe des zugehörigen Pixels berechnet. Mit diesem Verfahren können Effekte wie Reflektion, Brechung und Schattenwurf von Objekten sehr leicht durch rekursive Strahlverfolgung realisiert werden. Auch hier kann man mittels eines Quantenalgorithmus die Suche nach dem am nähesten liegenden Objekt durchführen.

- *Radiosity*: Hierbei handelt es sich um ein Verfahren, welches gut geeignet ist um indirekte Beleuchtung in einer Szene darzustellen. Die Grundidee basiert hier auf der Energieerhaltung, d.h. man berechnet die Lichtintensität als Summe des vom Objekt selbst emittierten Lichts und des Lichts, welches von den anderen Objekten der Szene auftrifft. Realisiert werden kann dieses Verfahren durch eine Erweiterung des Raytracers. Daher kann auch dieses Verfahren beschleunigt werden, indem man die Suche nach dem nähesten Objekt in einen Quantenalgorithmus auslagert.

In dieser Arbeit werden wir betrachten, wie der Faktorisierungsalgorithmus von Shor auf einem Quantencomputer realisiert werden kann und welche Hilfsmittel wir hierbei benötigen.
Im ersten Kapitel werden wir einige Grundbegriffe der Quantenmechanik einführen, die zum Verständnis der Vorgänge in einem Quantencomputer benötigt werden.

Im zweiten Kapitel führen wir *Quantenbits* und *Quantenregister* ein. Obendrein betrachten wir, wie wir Operationen auf Quantenbits durchführen können und wie wir mit einem Quantencomputer interagieren können. Zum Abschluss dieses Kapitels stellen wir schließlich einige Grundregeln zusammen, die wir bei der Konstruktion von Quantenalgorithmen beachten müssen.
Im dritten Kapitel betrachten wir zuerst zwei einfachere Algorithmen (ein Zufallsgenerator und der Algorithmus von Deutsch) und danach, wie wir mit Hilfe von Quantengattern arithmetische Operationen und die *Quantenfouriertransformation* realisieren können.
Das vierte Kapitel ist dann ganz dem *Algorithmus von Shor* gewidmet. Wir führen hier zunächst einige wichtige Hilfmittel aus der Zahlentheorie/Algebra ein (*euklidischer Algorithmus*, *Kettenbruchapproximation*) und analysieren danach zuerst den klassischen Teil des Faktorisierungsalgorithmus und dann den Quantenteil. Am Schluss dieses Kapitels betrachten wir die Komplexität des Quantenalgorithmus.
Das fünfte Kapitel gibt einen Einblick, wie man Quantenregister auf einem klassischen Rechner simulieren kann und wir betrachten einige klassische Algorithmen, die wir zur Durchführung des klassischen Teils des Algorithmus von Shor benötigen (schnelles Potenzieren, Primzahltest und Test ob eine Zahl eine Primzahlpotenz ist). In diesem Kapitel finden sich zudem noch einige Beispielmessungen, die mit Hilfe des Begleitprogramms zur Arbeit erstellt wurden.
Als sehr hilfreich für den Einstieg in die Welt der Quantencomputer hat sich das Buch „Quantum Computing verstehen" von Matthias Homeister erwiesen. Der Autor verlangt nur sehr wenige Vorkenntnisse und führt den Leser auf anschauliche Art und Weise durch die Grundlagen des Quantencomputers und gibt eine Einführung in Quantenalgorithmen wie den Grover-Algorithmus und auch den Algorithmus von Shor und erleichtert daher auch das Verständnis weiterführender Artikel. Obendrein bietet das Buch auch noch einen Einblick in das mit Quantencomputern sehr stark verwandte Thema der Quantenkryptographie.

Inhaltsverzeichnis

Kapitel 1

Grundprinzipien der Quantenmechanik

Inhaltsangabe

In diesem Kapitel behandeln wir die zur Beschreibung eines Quantencomputers benötigten Grundprinzipien der Quantenmechanik. Zuerst werden wir eine weit verbreitete Schreibweise zur Kennzeichnung quantenmechanischer Zustände kennenlernen. Danach werden wir die die Begriffe *Superposition* und *Verschränkung* näher betrachten. Dieses Kapitel orientiert sich zum größten Teil an [10] und [13].

1.1 Dirac Schreibweise

In der Quantenmechanik wird ein physikalisches System mit folgenden drei Bausteinen beschrieben:

- *Observable*: Messbare Größen eines Systems (z.B. Energie, Ort, Impuls)
- *Zustand*: Der Systemzustand wird durch eine sogenannte *Wellenfunktion* beschrieben. Diese Funktion legt fest, wie sich das System verhält und mit welcher Wahrscheinlichkeit welche Observable gemesen wird.
- *Dynamik*: Die zeitliche Entwicklung eines Systems. Diese wird durch Operatoren (z.B. *Hamiltonoperator*) festgelegt.

Die Wellenfunktion, die den Zustand des Systems charakterisiert, kann als Vektor in einem Hilbertraum aufgefasst werden. Als Basis betrachten wir Eigenfunktionen von geeigneten hermiteschen Operatoren (z.B. Hamiltonoperator in der Energiedarstellung, Ortsoperator in der Ortsdarstellung, Impulsoperator in der Impulsdarstellung). Wollen wir einen Zustand ψ unabhängig von seiner Darstellung angeben, so bezeichnen wir diesen mit $|\psi\rangle$. Diese Notation ist unter dem Namen *Dirac Notation* (bzw. *Bra-Ket-Notation*) bekannt und in der Quantenmechanik weit verbreitet.
Zustandsvektoren werden hierbei immer in der Form $|\psi\rangle$ geschrieben, während man die adjungierten Vektoren mit $\langle\psi|$ bezeichnet.
Die Multiplikation von $\langle a|$ mit $|b\rangle$ ergibt das Skalarprodukt der beiden Vektoren a und b, welches meist mit $\langle a|b\rangle$ bezeichnet wird. Nach dem englischen Wort **bra**c**ket** bezeichnet man deshalb die Zustandsvektoren als *Ket* und die dazu adjungierten Vektoren als *Bra*.

1.2 Superposition

Grundlegend für einen Quantencomputer ist die Tatsache, dass ein Teilchen welches den Gesetzen der Quantenmechanik folgt sich in einem sogenannten *Superpositionszustand* befinden kann.
Zur Veranschaulichung dieses Phänomens gibt es ein vom Physiker Erwin Schrödinger (1887-1961) vorgeschlagenes Gedankenexperiment, welches unter dem Namen „*Schrödingers Katze*" große Bekanntschaft erlangt hat:
In einer verschlossenen Kiste befindet sich ein instabiler Atomkern, der innerhalb einer gewissen Zeit mit einer bestimmten Wahrscheinlichkeit zerfällt. Sollte dies eintreten, registriert ein Geigerzähler den Zerfall und es strömt Giftgas aus, welches eine Katze tötet die sich ebenfalls im Raum aufhält. Da die Kiste jedoch verschlossen ist, wissen wir nicht genau ob die Katze noch lebt. Wir können nur sagen, dass die Katze mit einer gewissen Wahrscheinlichkeit tot oder lebendig ist. Bestimmen können wir den genauen Zustand erst, wenn wir die Kiste öffnen und nachsehen.
Eine gewissermaßen halbtote-halblebendige Katze ist schwer vorstellbar, jedoch ist dies in quantenmechanischen Systemen nicht ungewöhnlich. So können Elektronen z.B. sowohl als Welle als auch als Teilchen aufgefasst werden, Atome können mehrere Energieniveaus annehmen oder ein Photon kann beim Auftreffen auf eine Glasplatte transmittiert oder reflektiert werden. Diese Zustände treten, solange man das System nicht beobachtet, alle gleichzeitig mit entsprechenden Wahrscheinlichkeiten auf. Beobachtet man nun aber das System erkennt man nur eine eindeutige Messgröße und beeinflusst somit den Ablauf des Versuchs. Wir dürfen bei Schrödingers Katze also - bildlich gesprochen - nicht einen Glaskasten verwenden, den wir ständig beobachten können, da wir sonst zu jedem Zeitpunkt erkennen können, ob die Katze noch lebt. Eine Superposition kann sich somit nicht ausbilden.

1.3 Verschränkung

Ein weiteres für Quantencomputer wichtiges Phänomen ist die *Verschränkung*. Es handelt sich hierbei um die Eigenschaft, dass sich in einem quantenmechanischen System Teilchen gegenseitig beeinflussen können, obwohl sie räumlich voneinander getrennt sind. Wir werden z.B. sehen, dass die Messung eines Teilchens zur Folge hat, dass bei anderen Teilchen manche Zustände nicht mehr möglich sind. Dieser Effekt kann dazu benutzt werden um Quantenalgorithmen so zu konstruieren, dass am Schluss ein „*reiner Zustand*" (d.h. keine Superposition) entsteht. Die Messung dieses Zustands erlaubt uns dann mit Gewissheit Aussagen über ein oder mehrere Teilchen zu treffen (siehe z.B. Algorithmus 3.2.2).
Eine prominente Anwendung für verschränkte Zustände ist die sogenannte Quantenteleportation (siehe hierzu z.B.: [19] und [15]).

Kapitel 2

Theorie des Quantencomputers

Inhaltsangabe

In diesem Kapitel definieren wir *Quantenberechnungen* als abstraktes mathematisches Konzept, ohne auf physikalische Realisierbarkeit einzugehen. Wir werden die Grundregeln kennenlernen, die man bei jedem *Quantenalgorithmus* zu beachten hat und uns mit einigen ausgewählten *Operationen* beschäftigen. Dieses Kapitel richtet sich zum größten Teil nach [13] und [16].

2.1 Das Quantenbit

Um ein Bit in einem klassischen Rechner zu realisieren benötigt man zwei sauber definierte Zustände, die mit 0 und 1 bezeichnet werden.
In der TTL (Transistor Transistor Logik) sind dies typischerweise Spannungsbereiche von 50mV - 200mV (interpretiert als ,,0") und 3,5V - 5V (interpretiert als ,,1") (vgl. [14] S. 572).

Wir führen *Quantenbits* als Erweiterung der klassischen Bits ein.
Zur Realisierung von Quantenbits benötigen wir ein den Gesetzen der Quantenmechanik folgendes System, bei dem man zwei verschiedene und klar voneinander trennbare (und vor allem messbare) Zustände festlegen kann.
Diese werden als $|0\rangle$ und $|1\rangle$ bezeichnet (vergleiche auch Kapitel 1.1).
Ein Qubit $|x\rangle$ kann entweder eindeutig in einem reinem Zustand $|x\rangle = |i\rangle$ mit $i \in \{0, 1\}$ sein (dies entspricht dann einem klassischen Bit), oder als Superposition dieser Zustände vorliegen, z.B.

$$|x\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle.$$

Allgemein gilt:

Definition 2.1.1. *Ein* Quantenbit *(oder kurz:* Qubit*)* $|x\rangle$ *kann einen Zustand* $|x\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle$ *annehmen, wobei* $\alpha, \beta \in \mathbb{C}$ *mit*

$$|\alpha|^2 + |\beta|^2 = 1.$$

Analog zum Gedankenexperiment von Schrödingers Katze kann der Superpositionszustand nur bestehen, solange das Bit von der Außenwelt abgeschottet ist. Um Berechnungen anstellen zu können und Ergebnisse zu bekommen müssen wir jedoch auf das Quantenbit zugreifen können. Hierzu betrachten wir, was bei der Messung eines Quantenbits geschieht.
Befindet sich $|x\rangle$ vor der Messung in der Superposition $|x\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle$, dann zerstört die Messung die Superposition und man erhält

- mit Wahrscheinlichkeit $|\alpha|^2$ den Zustand $|0\rangle$,
- mit Wahrscheinlichkeit $|\beta|^2$ den Zustand $|1\rangle$.

$|0\rangle$ und $|1\rangle$ können als Basisvektoren eines zweidimensionalen Vektorraums aufgefasst werden, womit man festlegen kann:

$$|0\rangle := \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

$$|1\rangle := \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Somit ist:

$$|x\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle =: \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

2.2 Vom Qubit zum Quantenregister

Wie wir später sehen werden benötigen wir für die meisten Quantenalgorithmen mehr als nur ein einziges Quantenbit. Wir definieren in diesem Abschnitt was ein *Quantenregister* ist und wie es sich darstellen lässt.
Zuerst betrachten wir jedoch ein klassisches Register. Es handelt sich hierbei um eine Folge von Bits die jeweils die Zustände 0 und 1 annehmen können. In einem klassischen 2-Bit Register sind beispielsweise die Zustände 00, 01, 10 und 11 möglich. Diese Zustände kann man als Binärdarstellung ganzer Zahlen auffassen, in unserem Beispiel also 0, 1, 2 und 3.
Es ist somit möglich auf diese Weise mit einem n-Bit Register die Zahlen $0, 1, \ldots, 2^n - 1$ darzustellen.
Wie wir bereits gesehen haben kann sich ein Quantenbit im Gegensatz zu einem klassischen Bit in einer Überlagerung der beiden Zustände befinden. Betrachten wir nun beispielsweise ein Register mit zwei voneinander unabhängigen Quantenbits

$$|R\rangle := |x_1\rangle|x_0\rangle,$$

wobei

$$\begin{aligned} |x_0\rangle &= \gamma_0|0\rangle + \gamma_1|1\rangle, \\ |x_1\rangle &= \beta_0|0\rangle + \beta_1|1\rangle. \end{aligned}$$

Das komplette Register können wir als *Kroneckerprodukt* (auch bekannt als *Ternsorprodukt*) der beiden Zustände des Quantenbits definieren.

$$\begin{aligned} |R\rangle = |x_1\rangle|x_0\rangle &= \\ &= (\beta_0|0\rangle + \beta_1|1\rangle) \cdot (\gamma_0|0\rangle + \gamma_1|1\rangle) = \\ &= \beta_0\gamma_0|0\rangle|0\rangle + \beta_0\gamma_1|0\rangle|1\rangle + \beta_1\gamma_0|1\rangle|0\rangle + \beta_1\gamma_1|1\rangle|1\rangle =: \\ &=: \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \end{aligned}$$

Interpretieren wir die Indizes als Binärdarstellung ganzer Zahlen, so können wir ein *Quantenregister* wie folgt definieren:

Definition 2.2.1. *Ein n-Bit* Quantenregister $|R\rangle$ *ist eine Hintereinanderschaltung von n Quantenbits und kann sich in Zuständen der Form*

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

befinden, wobei $\alpha_i \in \mathbb{C}$ *und*

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

Eine Messung des Registers ergibt mit Wahrscheinlichkeit $|\alpha_i|^2$ *den reinen Zustand* $|i\rangle$.

Analog zu Quantenbits können wir die Zustände eines Quantenregisters mit n Bits als 2^n-dimensionalen Vektor auffassen. Die Basis bilden hierbei wieder die reinen Zustände $|0\ldots0\rangle, |0\ldots01\rangle, \ldots, |1\ldots1\rangle$.
Es gilt die Zuordnung:

$$|0\ldots0\rangle =: \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{C}^{2^n},$$

$$|0\ldots01\rangle =: \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{C}^{2^n},$$

$$\vdots$$

$$|1\ldots1\rangle =: \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \in \mathbb{C}^{2^n}.$$

Wie bereits in Abschnitt 1.3 angedeutet, können quantenmechanische Teilchen verschränkt sein. Diesen Effekt nutzt man häufig aus um spezielle Zustände im Quantenregister zu erzeugen. Es existiert z.B. die Möglichkeit ein 2-Bit Quantenregister auf die Form $|R\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ zu bringen. Misst man nun eines der beiden Quantenbits, so bricht der Superpositionszustand zusammen und man weiß ohne weitere Messung, dass sich das zweite Bit im gleichen Zustand befindet. Wir nennen das Quantenregister in diesem Fall *verschränkt* (engl.: *entangled*).
Verschränkte Quantenregister können nicht als Kroneckerprodukt mehrerer Quantenbits geschrieben werden, da die Quantenbits nach der Verschränkung nicht mehr voneinander unabhängig sind.

2.3 Operationen auf Qubits und Quantenregistern

In diesem Abschnitt werden wir uns mit zwei Klassen von Operationen auf Quantenregistern befassen.
Der erste Unterabschnitt befasst sich mit unitären Matrizen, mit denen man Operationen auf Quantenregistern beschreiben kann.
Im zweiten Unterabschnitt behandeln wir die erste Klasse von Operationen, die sich dadurch auszeichnen, dass sie nur auf einzelne Qubits wirken.

Im dritten Unterabschnitt lernen wir die Klasse der *kontrollierten Operationen* kennen. Dies sind Operationen, die nur auf ein Quantenbit wirken, wenn ein oder mehrere andere Quantenbits (sogenannte *Kontrollbits*) im Zustand $|1\rangle$ sind.

2.3.1 Unitäre Matrizen

Wie wir im vorherigen Kapitel gesehen haben können wir uns den Zustand eines n-Bit Quantenregisters als Einheitsvektor in $\mathbb{C}^{2^n}$ vorstellen. Um Quantenalgorithmen zu konstruieren müssen wir jedoch auch eine Möglichkeit haben den Zustand eines Registers ändern zu können.
Wir stellen an eine Operation auf einem Quantenregister folgende Forderungen:

1. Ein zulässiger Zustand soll wieder ein zulässiger Zustand werden. Wir müssen also ein Einheitsvektor wieder auf einen Einheitsvektor abbilden (*Längenerhaltung*).

2. Das Skalarprodukt zweier Zustände soll nicht verändert werden, damit die Bilder orthogonaler Vektoren wieder orthogonal sind (*Winkelerhaltung*).

Diese Forderungen führen uns zwangsläufig auf unitäre Abbildungen.

Definition 2.3.1. *Eine Operation auf n Quantenbits ist eine unitäre Abbildung* $U : \mathbb{C}^{2^n} \to \mathbb{C}^{2^n}$.

2.3.2 Operationen auf einem Qubit

Diese Klasse von Operationen kann man sich am besten in der Matrixschreibweise vorstellen. Es handelt sich hierbei um eine Multiplikation einer unitären Matrix mit dem Zustandsvektor des Quantenbits.
Ein wichtiger Vertreter dieser Klasse ist die *Hadamardtransformation*, welche ein Quantenbit von einem reinen Zustand $|0\rangle$ oder $|1\rangle$ in eine Superposition beider Zustände überführt und umgekehrt.
Sie wird durch folgende Vorschrift definiert:

$$|0\rangle \to \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle),$$
$$|1\rangle \to \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

In Matrixdarstellung kann die Hadamardtransformation definiert werden als

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

Lemma 2.3.2. *Die Hadamardtransformation ist eine unitäre Abbildung.*

Beweis. Da H nur reelle Einträge besitzt und symmetrisch ist, gilt

$$\bar{H}^T = H^T = H.$$

Damit folgt:

$$\bar{H}^T \cdot H = H^T \cdot H = H \cdot H = \frac{1}{2} \cdot \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^2 = \frac{1}{2} \cdot \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

□

Weitere wichtige Operationen auf Quantenregistern sind durch die *Pauli-Matrizen* gegeben:

$$X := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$
$$Y := \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix},$$
$$Z := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

2.3.3 Kontrollierte Operationen

Bei der zweiten Klasse von Operationen handelt es sich um die *kontrollierten Operationen*. Ein typisches Einsatzgebiet dieser Operationen ist die Nachbildung klassischer Bitoperatoren auf einem Quantencomputer. Betrachten wir das *klassische exklusive Oder* (kurz *XOR*), dargestellt durch die in Tabelle 2.1 angegebene Rechenvorschrift, sehen wir, dass diese Abbildung nicht umkehrbar ist, da $0 \oplus 0 = 0 = 1 \oplus 1$ und $1 \oplus 0 = 1 = 0 \oplus 1$.

$\oplus$	0	1
0	0	1
1	1	0

Tabelle 2.1: Rechenvorschrift für das ,,exklusive Oder".

Da aber, wie in Abschnitt 2.3 gezeigt, Operationen auf Quantenregistern umkehrbar sein müssen, ersetzen wir diese Abbildung durch eine umkehrbare Version, indem wir uns zusätzlich zum Ergebnis der Operation den ersten Operanden merken.

Lemma 2.3.3. *Eine unitäre Version des „exklusiven Oder", auch bekannt als* kontrolliertes exklusives Oder *bzw.* CNOT*, ist durch die Abbildung*

$$|x\rangle|y\rangle \to |x\rangle|y \oplus x\rangle$$

gegeben.

Beweis. Für die Abbildung gilt:

$$|0\rangle|0\rangle \to |0\rangle|0 \oplus 0\rangle = |0\rangle|0\rangle,$$

$$|0\rangle|1\rangle \to |0\rangle|1 \oplus 0\rangle = |0\rangle|1\rangle,$$

$$|1\rangle|0\rangle \to |1\rangle|0 \oplus 1\rangle = |1\rangle|1\rangle,$$

$$|1\rangle|1\rangle \to |1\rangle|1 \oplus 1\rangle = |1\rangle|0\rangle.$$

Die Abbildung vertauscht demnach $|1\rangle|0\rangle$ mit $|1\rangle|1\rangle$ und lässt die anderen beiden Zustände unverändert. Die Bilder der Abbildung bleiben unterscheidbar. Erneutes Anwenden der Abbildung tauscht die Komponenten wieder zurück, weshalb in diesem Fall die Abbildung mit ihrer Umkehrabbildung übereinstimmt. Da die Abbildung obendrein mit einer symmetrischen, reellwertigen Matrix dargestellt werden kann, ist sie unitär. □

Eine wichtige Eigenschaft *kontrollierter Operationen* ist, dass sie nur wirken, falls das *Kontrollbit* im Zustand $|1\rangle$ ist. Im obigen Fall wirkt also die Pauli-Matrix X auf das zweite Quantenbit, wenn das erste Bit im Zustand $|1\rangle$ ist. Falls das Kontrollbit $|0\rangle$ ist, wird das Register nicht geändert. Mit kontrollierten Operationen können somit, wie wir später noch sehen werden, Bedingungen in Quantenalgorithmen formuliert werden.

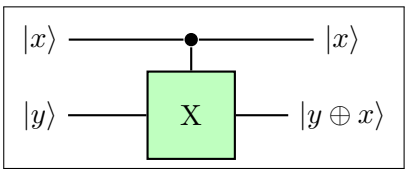

Abbildung 2.1: Symbol für das kontrollierte XOR.

Eine weitere wichtige Operation für Quantencomputer stellt das *Toffoligatter* dar, da dies, wie z.B. das NAND-Gatter bei klassischen Computern, ein universelles Gatter ist.
Mit Hilfe dieser Operation ist es möglich klassische Schaltungen auf einem Quantencomputer umzusetzen. Wir werden aber später noch sehen, dass dieses Gatter auch sehr wichtig ist um arithmetische Operationen auf Quantencomputern durchführen zu können.

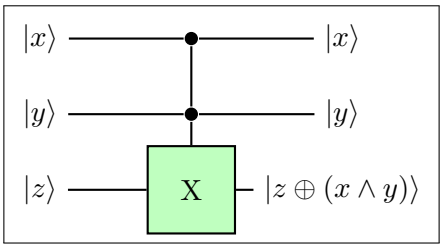

Abbildung 2.2: Symbol für das Toffoligatter.

Lemma 2.3.4. *(Toffoligatter) Das Toffoligatter beschreibt eine unitäre Abbildung.*

Beweis. Der Beweis hierfür funktioniert analog zum Beweis von Lemma 2.3.3. □

2.4 Einfache Rechenschritte

Nachdem wir einige Operationen kennengelernt haben stellt sich die Frage wie der Aufwand von Quantenalgorithmen abgeschätzt werden kann. Hierzu müssen wir definieren was *einfache Rechenschritte* auf einem Quantencomputer sind.
Wir werden sehen, dass wir unsere Quantenalgorithmen mit Hilfe von Gattern konstruieren können bei denen höchstens 3 Quantenbits beteiligt sind, weshalb wir einen einfachen Rechenschritt wie folgt definieren:

Definition 2.4.1. *Ein* einfacher Rechenschritt *auf einem Quantencomputer ist eine unitäre Transformation an der höchstens drei Bits beteiligt sind.*

Um den Aufwand einer Registeroperation abzuschätzen, muss man diese also in Drei-Bit-Transformationen zerlegen.

2.5 Interaktion mit Quantencomputern

Bisher haben wir uns nur damit beschäftigt, wie wir Daten in einem Quantencomputer speichern können und wie wir sie manipulieren können. In diesem Kapitel betrachten wir, wie wir mit einem Quantencomputer interagieren können, das heißt primär: Wie können wir Daten eingeben und ausgeben.
Zu beachten ist immer, dass jegliche Interaktion mit einem Quantencomputer damit verbunden ist, dass wir das quantenmechanische System verändern. Wir dürfen also nur an gewissen Stellen in die Algorithmen eingreifen und müssen uns der Folgen für den Algorithmus stets bewusst sein.

2.5.1 Eingabe von Daten

Um Daten in den Quantencomputer einzugeben haben wir mehrere Möglichkeiten. Eine davon ist, dass wir den Zustand des Quantenregisters zu Beginn des Algorithmus steuern können.
Nehmen wir an wir betrachten den Kernspin als Quantenbit (siehe auch Anhang C.3), so können wir durch eine Messung des Spins vor dem eigentlichen Algorithmus sicherstellen, dass das Teilchen im Zustand $|\uparrow\rangle$ oder $|\downarrow\rangle$ ist. Lassen wir den Algorithmus ablaufen, starten wir mit genau einer dieser beiden Möglichkeiten, da eine Superposition beider Zustände aufgrund der Messung nicht mehr möglich ist.
Eine zweite Möglichkeit die Eingabe des Algorithmus zu steuern stellt der Versuchsaufbau an sich dar. Wie wir sehen werden, können manche Schritte eines Quantenalgorithmus auf klassische Rechner ausgelagert werden. Die dort erhaltenen Resultate steuern den Ablauf des Quantenalgorithmus, indem man die Schaltung passend zu diesem Ergebnis konstruiert.
Eine weitere Möglichkeit zur Steuerung des Algorithmus und der Eingabe von Daten besteht darin, dass man an gewissen Stellen im Algorithmus das Register - oder zumindest Teile davon - misst. Das Ergebnis ist in vielen Fällen ohne weitere Bedeutung, jedoch bewirkt dies, dass ein Superpositionszustand soweit zusammenfällt, dass er dem Messergebnis entspricht.
Das letzte Mittel bietet eventuell eine gute Möglichkeit um eine gewisse Fehlerkorrektur im Algorithmus durchzuführen. Wir werden bei der Konstruktion arithmetischer Operationen auf einem Quantencomputer Stellen sehen, an der man eine Messung einbauen könnte deren Ergebnis scheinbar nicht weiter von Bedeutung ist. Jedoch wissen wir dort, dass ein Teil des Registers in einem bestimmten Zustand sein sollte. Finden wir nun bei der Messung einen anderen Wert, wissen wir, dass ein Fehler im Algorithmus aufgetreten sein muss und können abbrechen (siehe hierzu Bemerkung 3.3.2). Falls wir den wahren Wert gemessen haben wissen wir zumindest, dass die bisherigen Schritte plausibel waren und erreichen aber, dass Zustände, die nicht zu dem Ergebnis passen, zerstört werden.

2.5.2 Ausgabe von Daten

Die Datenausgabe in einem Quantencomputer erfolgt grundsätzlich über Messungen. Bei einer *Messung* erhalten wir immer genau einen der Zustände in denen sich das Quantenregister zur Zeit befindet. Eine Superposition ist nicht erkennbar.
Ziel von Quantenalgorithmen ist es das Quantenregister in einen Zustand zu versetzen, so dass bestenfalls am Schluss nur ein eindeutiges Ergebnis entsteht (vgl. Algorithmus 3.2.2) oder man zumindest mit hoher Wahrscheinlichkeit mit dem Messergebnis den gewünschten Wert ermitteln kann.
Mathematisch gesehen stellt eine Messung die Projektion des Zustandsvek-

tors auf einen Unterraum dar. Wir fassen den Zustand eines n-Bit Quantenregisters als Vektor im $\mathbb{C}^{2^n}$ auf. Dieser wird von den Basisvektoren $|0\rangle$, $|1\rangle$, ..., $|2^n - 1\rangle$ aufgespannt. Messen wir nun m Quantenbits im Register, so sind deren Zustände festgelegt und das Quantenregister befindet sich in dem zum gemessenen Zustand passenden 2^{n-m}-dimensionalen Unterraum.

Beispiel 2.5.1. *Messen eines Quantenbits im Quantenregister*

$$|R\rangle = |x_0\rangle|x_1\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

liefert folgende möglichen Fälle:

- $|x_0\rangle = |0\rangle \Rightarrow |R\rangle \in span\{|00\rangle, |01\rangle\}$,
- $|x_0\rangle = |1\rangle \Rightarrow |R\rangle \in span\{|10\rangle, |11\rangle\}$,
- $|x_1\rangle = |0\rangle \Rightarrow |R\rangle \in span\{|00\rangle, |10\rangle\}$,
- $|x_1\rangle = |1\rangle \Rightarrow |R\rangle \in span\{|01\rangle, |11\rangle\}$.

2.6 Grundregeln für Quantenalgorithmen

Zusammenfassend stellen wir einige Grundregeln auf, die Quantenalgorithmen berücksichtigen müssen (vgl. auch [16] Kap. 1.2).

Grundregel 2.6.1. *Ein Quantenbit ist die kleinste Informationseinheit in einem Quantenregister. Klassische Bits sind ein Spezialfall von Quantenbits.*

Grundregel 2.6.2. *Quantenalgorithmen sind probabilistisch.*

Grundregel 2.6.3. *Messungen zerstören eine Superposition quantenmechanischer Zustände.*

Grundregel 2.6.4. *In einem n-Bit Quantenregister können wir mit 2^n verschiedenen Zuständen rechnen. Der nutzbare Bereich ist daher exponentiell größer als der eines klassischen Rechners.*

Grundregel 2.6.5. *Operationen auf einem Quantencomputer müssen umkehrbar sein um eine Superposition nicht zu zerstören.*

Grundregel 2.6.6. *Quantencomputer ermöglichen* massive Parallelisierung. *Ein Rechenschritt auf einem n-Bit Register kann gleichzeitig 2^n Werte ändern.*

Grundregel 2.6.7. *Quanteninformation kann nicht kopiert werden ohne den Ausgangszustand zu verändern (*no-cloning-theorem*).*

Grundregel 2.6.8. *Quantenregister sind immer im Zustand $|0\rangle$ initialisiert. Wollen wir einen Quantenalgorithmus mit einem Register starten, das sich in einem anderen Zustand befindet, müssen wir Diesen zuerst mit passenden Quantenoperation erzeugen.*

Kapitel 3

Quantenalgorithmen

Inhaltsangabe

In diesem Kapitel widmen wir uns ersten *Quantenalgorithmen*. Zuerst betrachten wir einen *Zufallsgenerator*, mit dessen Hilfe wir die Wirkung von Hadamardgattern und der Messung betrachten und danach den *Algorithmus von Deutsch*. Hier werden wir sehen, wie wir aus einer klassischen Funktion einen Quantenalgorithmus bauen können und wie man durch *Quantenparallelismus* Vorteile gegenüber klassischen Verfahren hat. Danach betrachten wir, wie wir mit den in den vorherigen Kapiteln angesprochenen Quantengattern arithmetische Operationen wie Addition, Multiplikation und *Exponentiation* durchführen können. Am Schluss widmen wir uns der *Quantenfouriertransformation*, die wir im Algorithmus von Shor benötigen.

3.1 Ein Zufallsgenerator

Algorithmus 3.1.1. *Zufallsgenerator*
Eingabe: $n \in \mathbb{N}$ *(Anzahl der Bits).*
Ausgabe: *Eine zufällige Zahl zwischen* 0 *und* $2^n - 1$

1. *Lege ein n-Bit Quantenregister $|R\rangle$ im Zustand $|\underbrace{0\ldots0}_{n}\rangle$ an.*

2. *Wende die Hadamardtransformation auf alle Quantenbits an.*

3. *Miss das Register $|R\rangle$ und gib das Ergebis aus.*

Im ersten Schritt wird das Quantenregister in den reinen Zustand $|0\rangle$ versetzt. Wendet man auf jedes Bit des Registers die Hadamardtransformation an wird es in eine Superposition aller möglichen Zustände versetzt, also ausgeschrieben:

$$H_n|R\rangle = \left(\frac{1}{\sqrt{2}}\right)^n \cdot \sum_{i=0}^{2^n-1} |i\rangle.$$

Misst man dieses Register erhält man eine Zahl zwischen 0 und $2^n - 1$.

Anzumerken ist, dass es sich hier um einen *echten Zufallsgenerator* handelt und keinen *Pseudozufallsgenerator* wie bei klassischen Rechnern.

3.2 Algorithmus von Deutsch

Als zweites Beispiel betrachten wir den *Algorithmus von Deutsch* (vgl. [13] Kapitel 2.6 und [8] Abschnitt 3).

Es handelt sich hierbei um einen Algorithmus mit dem man untersuchen kann, ob eine Funktion $f : \{0, 1\} \to \{0, 1\}$ konstant ist oder nicht. Auf einem klassischen Computer müssen wir $f(0)$ und $f(1)$ berechnen um eine Aussage darüber treffen zu können. Mit Hilfe eines Quantencomputers können wir dies entscheiden, indem wir nur eine Messung durchführen.

Das Ziel ist eine unitäre Operation zu finden, mit deren Hilfe wir die gegebene Funktion auswerten können.

Um zu gewährleisten, dass die Operation umkehrbar ist, führen wir (ähnlich wie bei CNOT) ein Kontrollbit ein. Wir definieren uns somit eine Abbildung:

$$U_f : |x\rangle|y\rangle \longmapsto |x\rangle|y \oplus f(x)\rangle.$$

Für die Funktion f existieren vier verschiedene Möglichkeiten:

$$f(x) = \begin{cases} 0 & x = 0 \\ 0 & x = 1, \end{cases} \qquad (I)$$

$$f(x) = \begin{cases} 0 & x = 0 \\ 1 & x = 1, \end{cases} \qquad (II)$$

$$f(x) = \begin{cases} 1 & x = 0 \\ 0 & x = 1, \end{cases} \qquad (III)$$

$$f(x) = \begin{cases} 1 & x = 0 \\ 1 & x = 1. \end{cases} \qquad (IV)$$

Diese vier Fälle setzen wir nun in Quantenalgorithmen um.

(I) $|x\rangle|y\rangle \longmapsto |x\rangle|y \oplus 0\rangle = |x\rangle|y\rangle,$

(II) $\begin{cases} |0\rangle|y\rangle \longmapsto |0\rangle|y \oplus 0\rangle = |0\rangle|y\rangle \\ |1\rangle|y\rangle \longmapsto |1\rangle|y \oplus 1\rangle, \end{cases}$

(III) $\begin{cases} |0\rangle|y\rangle \longmapsto |0\rangle|y \oplus 1\rangle \\ |1\rangle|y\rangle \longmapsto |1\rangle|y \oplus 0\rangle = |1\rangle|y\rangle, \end{cases}$

(IV) $|x\rangle|y\rangle \longmapsto |x\rangle|y \oplus 1\rangle.$

Somit ergibt sich:

Lemma 3.2.1. *Eine Funktion $f : \{0,1\} \to \{0,1\}$ kann als Quantenversion dargestellt werden durch*

$$f : |x\rangle|y\rangle \longmapsto U_f \cdot |x\rangle|y\rangle$$

wobei:

(I) $U_f := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$

(II) $U_f := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$

$$\textit{(III)}\ U_f := \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\textit{(IV)}\ U_f := \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Algorithmus 3.2.2. *Algorithmus von Deutsch*
Eingabe: *Eine Funktion $f : \{0,1\} \to \{0,1\}$, die als Quantenversion (vgl. Lemma 3.2.1) vorliegt.*
Ausgabe: *Die Funktion ist konstant, falls Messung den Wert 1 ergibt. Die Funktion ist nicht konstant, falls Messung den Wert 3 ergibt.*

1. *Lege ein Quantenregister mit 2 Bit im Zustand $|0\rangle|1\rangle$ an.*
2. *Wende die Hadamardtransformation auf beide Bits an.*
3. *Werte die Funktion f aus.*
4. *Wende die Hadamardtransformation erneut auf beide Bits an.*
5. *Messen des Registers und Ausgabe des Ergebnisses.*

Bevor wir zur Analyse des Algorithmus kommen, betrachten wir eine Idee des Algorithmus noch näher, die uns auch im Algorithmus von Shor begegnen wird.
Die Anwendung der Hadamardtransformation kann als Basiswechsel aufgefasst werden

$$|0\rangle \to |+\rangle := \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle),$$
$$|1\rangle \to |-\rangle := \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

In dieser neuen Basis werten wir die Funktion aus und führen anschließend wieder einen Basiswechsel zur ursprünglichen Basis durch. Damit erhalten wir ein Ergebnis, welches unsere Funktion charakterisiert.
Wir analysieren nun die einzelnen Schritte des Algorithmus:

1. Wir starten mit einem Quantenregister im Grundzustand $|0\rangle|0\rangle$. Anwenden des Pauli-X-Gatters (vgl. Abschnitt 2.3.2) auf das zweite Quantenbit erzeugt den gewünschten Anfangszustand.

2. Die Hadamardtransformation führt den bereits angesprochenen Basiswechsel durch:

$$H_2|0\rangle|1\rangle = |+\rangle|-\rangle = \frac{1}{2}(|0\rangle|0\rangle - |0\rangle|1\rangle + |1\rangle|0\rangle - |1\rangle|1\rangle).$$

3. Anwenden der Funktion $f : |x\rangle|y\rangle \longmapsto |x\rangle|y \oplus f(x)\rangle$ erzeugt den Zustand

$$\begin{aligned} U_f|+\rangle|-\rangle = \frac{1}{2}\big(&|0\rangle|0 \oplus f(0)\rangle - |0\rangle|1 \oplus f(0)\rangle + \\ &+ |1\rangle|0 \oplus f(1)\rangle - |1\rangle|1 \oplus f(1)\rangle\big) = \\ &= \frac{1}{2}\big(|0\rangle|f(0)\rangle - |0\rangle|1 \oplus f(0)\rangle + |1\rangle|f(1)\rangle - |1\rangle|1 \oplus f(1)\rangle\big) = \\ &= \frac{1}{2}\Big(|0\rangle \cdot \big(|f(0)\rangle - |1 \oplus f(0)\rangle\big) + |1\rangle \cdot \big(|f(1)\rangle - |1 \oplus f(1)\rangle\big)\Big). \end{aligned}$$

Es gilt

$$|f(x)\rangle - |1 \oplus f(x)\rangle = (-1)^{f(x)} \cdot (|0\rangle - |1\rangle)$$

und daraus ergibt sich

$$U_f|+\rangle|-\rangle = \frac{1}{2}\left((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\right) \cdot (|0\rangle - |1\rangle).$$

Man erkennt:

- Mit $f(0) = f(1) = \alpha$ gilt

$$\begin{aligned} U_f|+\rangle|-\rangle &= (-1)^{\alpha} \cdot \frac{1}{2} \cdot (|0\rangle + |1\rangle)(|0\rangle - |1\rangle) = \\ &= (-1)^{\alpha} \cdot |+\rangle|-\rangle. \end{aligned}$$

- Mit $f(0) \neq f(1)$ gilt

$$\begin{aligned} \Rightarrow U_f|+\rangle|-\rangle &= (-1)^{f(0)} \cdot \frac{1}{2} \cdot (|0\rangle - |1\rangle)(|0\rangle - |1\rangle) = \\ &= (-1)^{f(0)}|-\rangle|-\rangle. \end{aligned}$$

4. Nun wenden wir die Hadamardtransformation erneut an, was dazu führt, dass der Basiswechsel wieder rückgängig gemacht wird, d.h.

$$\begin{aligned} |+\rangle &\to |0\rangle, \\ |-\rangle &\to |1\rangle. \end{aligned}$$

- Ist f konstant, versetzen wir das Quantenregister in den Zustand $\pm|0\rangle|1\rangle$.
- Ist f nicht konstant, so erhalten wir den Zustand $\pm|1\rangle|1\rangle$.

5. Da das Register in beiden Fällen in einem reinen Zustand ist, können wir anhand des Messergebnisses bestimmen welcher Fall vorliegt. den Zustand $|0\rangle|1\rangle$ fassen wir als Messergebnis 1 auf, den Zustand $|1\rangle|1\rangle$ als Messergebnis 3.

Bemerkung 3.2.3. *Bei der Interpretation des Messergebnisses wurde die Reihenfolge der Quantenbits so festgelegt, dass sie der Lesereihenfolge einer Zahl in Binärdarstellung entsprechen wobei das niedrigste Quantenbit hier links steht. Im Begleitprogramm zu dieser Arbeit ist die Reihenfolge der Quantenbits jedoch vertauscht, d.h. das niedrigste Quantenbit steht rechts. Daher ist das Register zum Schluss im Zustand $|2\rangle$, falls die Funktion konstant war.*

3.3 Arithmetische Operationen

Wie wir bereits in Kapitel 2 gesehen haben, kann sich ein Quantenregister in einer Superposition mehrerer Zustände befinden. Ziel in den folgenden Abschnitten ist die Konstruktion eines Algorithmus, der diesen Zustand zur parallelen Berechnung von Potenzen $x^k \pmod n$ nutzen kann, wobei $x, n \in \mathbb{N}$ und $k \in \{0, 1, ..., 2^n - 1\}$ ist.
Wir werden im Folgenden, aufgrund der Komplexität des Verfahrens, nur die Grundideen zur Realisierung arithmetischer Operationen auf Quantenregistern - soweit diese für den Algorithmus von Shor nötig sind - näher behandeln. Eine ausführlichere Behandlung aller Schritte und die Analyse der verwendeten Quantengatter findet man in [4], [3] und [25].

3.3.1 Grundlegende Algorithmen

Wie bereits in der Einleitung zu diesem Abschnitt angedeutet, ist das Ziel dieses Abschnitts die Konstruktion eines Algorithmus zur Berechnung der Potenzen $x^0 \pmod n, x^1 \pmod n, x^2 \pmod n, ..., x^{2^n-1} \pmod n$, wobei wir die Parallelisierung ausnutzen wollen, die uns ein Quantencomputer bieten kann.
Wie auch auf einem klassischen Rechner führen wir diese Operation nicht direkt durch, sondern führen sie auf die Multiplikation und Addition zurück. Die Grundoperation die wir durchführen müssen ist die Addition modulo n. Wollen wir eine klassische Zahl $y < n$ zu einer Quantenzahl b addieren, d.h. b entspricht allen Zahlen die in der Superposition des Quantenregisters auftreten, so können wir dies als eine Funktion f auffassen mit

$$f(y) = \begin{cases} b + y & \text{falls } n - y > b \\ b + (y - n) & \text{falls } n - y \leq b. \end{cases}$$

Eine offensichtliche Schwierigkeit ist, dass wir eine klassische Zahl mit einer Quantenzahl vergleichen müssen, d.h. wir müssen im Quantenalgorithmus

entscheiden, welche der beiden Additionen durchzuführen ist. Ein weiteres Problem ist, dass wir sicherstellen müssen, dass die Operation umkehrbar ist um sie auf einem Quantencomputer durchführen zu können. Diesen Problemen widmen wir uns in Abschnitt 3.3.2.
Die Addition können wir noch zusätzlich mit einem Kontrollbit ausstatten (vgl. Abschnitt 2.3.3) und somit die Multiplikation mittels folgender Idee realisieren:
Sei

$$b = \sum_{i=0}^{K-1} b_i \cdot 2^i \text{ mit } b_i \in \{0,1\},$$

dann können wir schreiben:

$$b \cdot y \pmod n = \sum_{i=0}^{K-1} b_i \cdot (2^i \cdot y \pmod n).$$

y und n sind hierbei klassische Zahlen, die bereits vor Beginn des Quantenalgorithmus festgelegt sind.
Definieren wir $c_i := y \cdot 2^i \pmod n$, so können wir das Produkt berechnen

$$b \cdot y \pmod n = \sum_{i=0}^{K-1} b_i \cdot c_i \pmod n = \sum_{\substack{i=0 \\ b_i=1}}^{K-1} c_i \pmod n$$

Auf ähnliche Weise, wie wir die Multiplikation auf die Addition zurückgeführt haben, können wir auch das Potenzieren auf die Multiplikation zurückführen. Hierzu benutzen wir die *binäre Exponentiation* (engl. *repeated squaring* oder *square-and-multiply*, vergleiche auch Algorithmus 5.2.1).
Sei

$$k = \sum_{i=0}^{L-1} k_i \cdot 2^i \text{ mit } k_i \in \{0,1\},$$

dann ist

$$x^k = x^{\sum_{i=0}^{L-1} k_i 2^i} = \prod_{i=0}^{L-1} \left(x^{2^i}\right)^{k_i} = \prod_{\substack{i=0 \\ k_i=1}}^{L-1} x^{2^i}.$$

Auch hier können wir, da x eine klassische Zahl ist, wieder Vorbereitungen treffen und definieren $d_i := x^{2^i} \pmod n$.
Somit erhalten wir

$$x^k \pmod n = \prod_{\substack{i=0 \\ k_i=1}}^{L-1} d_i \pmod n.$$

3.3.2 Umsetzung auf einem Quantencomputer

Nachdem wir in Abschnitt 3.3.1 eine Strategie entwickelt haben, wie wir den Algorithmus umsetzen können, wenden wir uns in diesem Kapitel der Implementierung auf einem Quantencomputer zu.
Als erstes Hilfsmittel müssen wir einen Volladdierer konstruieren, welcher aus drei Eingabebits a, b und c die Summe s und ein Übertragsbit c' berechnet.

$$s = a \oplus b \oplus c, \tag{3.1}$$
$$c' = (a \wedge b) \vee (c \wedge (a \vee b)). \tag{3.2}$$

Da bei unserer Addition immer ein klassisches Bit a beteiligt ist, konstruieren wir zwei Operationen in Abhängigkeit von a. Hierzu benötigen wir zusätzlich zu $|b\rangle$ und $|c\rangle$ noch ein weiteres Quantenbit:

$$FA(a=0) : |b\rangle|c\rangle|0\rangle \longmapsto |b\rangle|b \oplus c\rangle|b \wedge c\rangle,$$
$$FA(a=1) : |b\rangle|c\rangle|0\rangle \longmapsto |b\rangle|b \oplus c \oplus 1\rangle|b \vee c\rangle.$$

Um die Multiplikation effizient durchführen zu können konstruieren wir einen *Volladdierer*, der zwei klassische Bits entgegennimmt und anhand eines Kontrollbits entscheidet, welches der Beiden verwendet wird. Diese Operation hat die Gestalt

$$FA(a, a') : |l\rangle|b\rangle|c\rangle|0\rangle \longmapsto |l\rangle|b\rangle|s\rangle|c'\rangle,$$

wobei s und c' wie in (3.1), (3.2) berechnet werden mit

$$a = \begin{cases} a & \text{falls } l = 0 \\ a' & \text{falls } l = 1. \end{cases}$$

Zusätzlich zum Volladdierer benötigen wir einen *Halbaddierer*. Dieser unterscheidet sich vom Volladdierer nur dadurch, dass das Übertragsbit nicht berechnet wird,

$$HA(a, a') : |l\rangle|b\rangle|c\rangle \longmapsto |l\rangle|b\rangle|s\rangle.$$

Haben wir Halb- und Volladdierer implementiert, können wir sie zu einem Addierer zusammensetzen, indem wir $K-1$ Volladdierer und einen Halbaddierer miteinander verketten, wobei K die Anzahl der Quantenbits im Register ist, aus dem die Quantenzahl gelesen wird. Wir erhalten

$$MADD(a, a') : |b\rangle|0\rangle|l\rangle \longmapsto |b\rangle|s\rangle|l\rangle,$$

wobei a, a', b Zahlen mit K-Bits sind und

$$s = b + \begin{cases} a' & \text{falls } l = 1 \\ a & \text{falls } l = 0 \end{cases} \pmod{2^K}.$$

Der hier vorgestellte Addierer kann mittels Toffoligattern, kontrollierten XOR und Pauli-X-Gattern gebaut werden (siehe hierzu [4] S. 24 ff.). Wir spendieren unserer Schaltung obendrein noch ein zweites Kontrollbit $|\tilde{l}\rangle$, da wir den Addierer bei Bedarf auch deaktivieren können wollen.
Wir konstruieren zuerst die Quantenschaltkreise der Halb- und Volladdierer, die zu den vier möglichen Eingaben passen. Die folgenden Diagramme sind von links nach rechts zu lesen, wobei wir, um Übersicht bewahren zu können, nach jedem Gatter den neuen Zustand des Quantenregisters angeben werden:

- $a = a' = 0$:

  Beide Eingabebits sind gleich, weshalb wir nur eine Schaltung konstruieren müssen, welche 0 zur aktuellen Eingabe addiert.

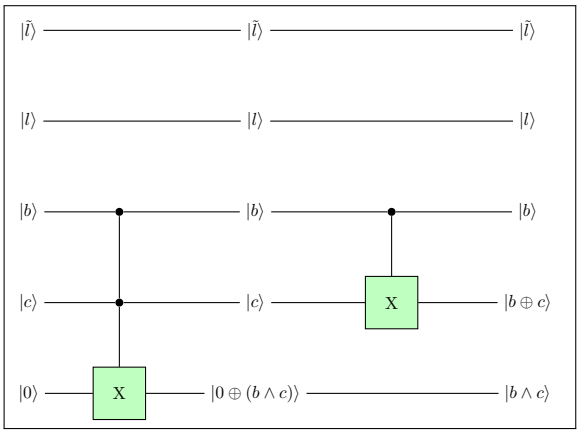

Abbildung 3.1: Volladdierer im Fall $a = a' = 0$.

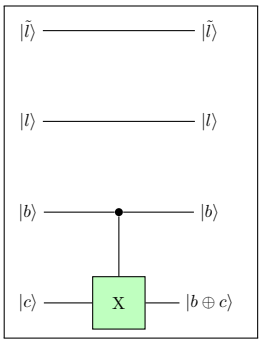

Abbildung 3.2: Halbaddierer im Fall $a = a' = 0$.

- $a = a' = 1$

  Auch hier sind die beiden Bits gleich, wir müssen allerdings sicherstellen, dass 1 addiert wird, falls $|\tilde{l}\rangle$ gesetzt ist und 0, falls $|\tilde{l}\rangle$ nicht gesetzt

ist. Da es sich hier um eine etwas aufwendigere operation handelt, definieren wir:

$$c_1 := (c \wedge \tilde{l}) \oplus ((c \oplus \tilde{l}) \wedge b)$$

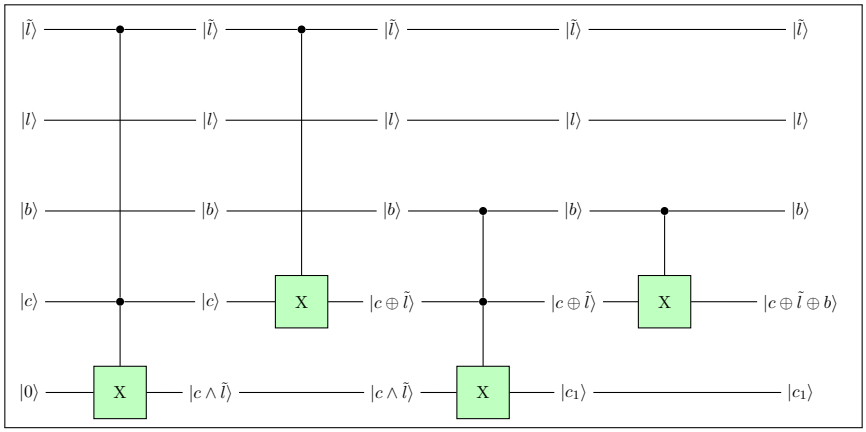

Abbildung 3.3: Volladdierer im Fall $a = a' = 1$.

Falls $|\tilde{l}\rangle = 0$ ist, erhalten wir wie gewünscht

$$\begin{aligned}|\tilde{l}\rangle|l\rangle|b\rangle|c\rangle|s\rangle \longmapsto |\tilde{l}\rangle|l\rangle|b\rangle|b \oplus 0 \oplus c\rangle|c_1\rangle = \\ = |\tilde{l}\rangle|l\rangle|b\rangle|b \oplus c\rangle|0 \oplus (c \oplus 0) \wedge b\rangle = \\ = |\tilde{l}\rangle|l\rangle|b\rangle|b \oplus c\rangle|c \wedge b\rangle.\end{aligned}$$

Ist $|\tilde{l}\rangle = 1$, so wird das Register folgendermaßen verändert:

$$\begin{aligned}|\tilde{l}\rangle|l\rangle|b\rangle|c\rangle|s\rangle \longmapsto |\tilde{l}\rangle|l\rangle|b\rangle|b \oplus 1 \oplus c\rangle|c_1\rangle = \\ = |\tilde{l}\rangle|l\rangle|b\rangle|b \oplus c \oplus 1\rangle|(c \wedge 1) \oplus (c \oplus 1) \wedge b\rangle = \\ = |\tilde{l}\rangle|l\rangle|b\rangle|b \oplus c\rangle|(c \wedge 1) \oplus (\neg c \wedge b)\rangle \overset{(*)}{=} \\ = |\tilde{l}\rangle|l\rangle|b\rangle|b \oplus c\rangle|(c \wedge 1) \vee (1 \wedge b)\rangle = \\ = |\tilde{l}\rangle|l\rangle|b\rangle|b \oplus c\rangle|(c \wedge (1 \vee b)) \vee (1 \wedge b)\rangle.\end{aligned}$$

Die Identität in $(*)$ lässt sich durch einsetzen aller vier möglichen Kombinationen nachprüfen.

Den Halbaddierer realisieren wir mit folgendem Schaltkreis:

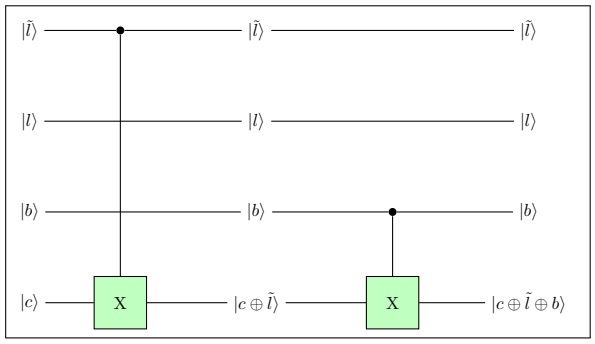

Abbildung 3.4: Halbaddierer im Fall $a = a' = 1$.

- $a = 0, a' = 1$

  Diese Schaltung gestaltet sich etwas komplizierter als die bisherigen Fälle, da wir hier beide Kontrollbits beachten müssen. Wir geben im Volladdierer daher nur die Operationen an und diskutieren die Änderungen anschließend.

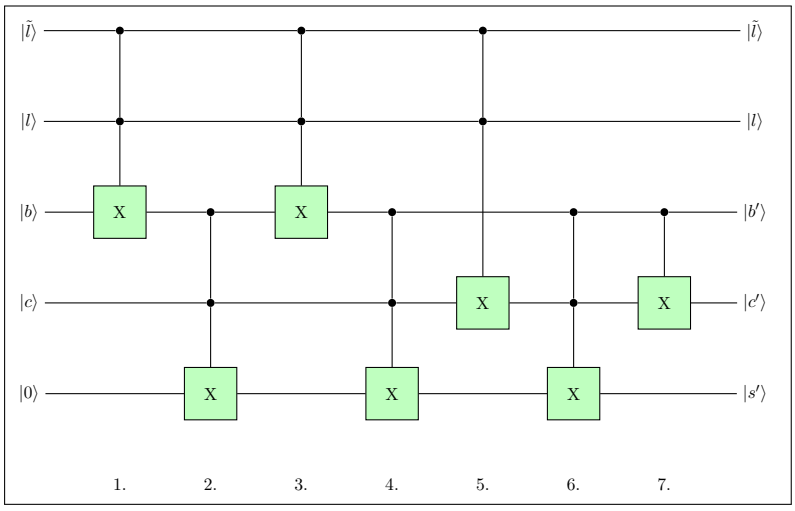

Abbildung 3.5: Volladdierer im Fall $a = 0, a' = 1$.

Der Zustand des Registers entwickelt sich während des Algorithmus wie folgt:

1. $|b\rangle \to |b \oplus (\tilde{l} \wedge l)\rangle$
2. $|0\rangle \to |0 \oplus ((b \oplus (\tilde{l} \wedge l)) \wedge c)\rangle = |(b \oplus (\tilde{l} \wedge l)) \wedge c\rangle$
3. $|b \oplus (\tilde{l} \wedge l)\rangle \to |b \oplus (\tilde{l} \wedge l) \oplus (\tilde{l} \wedge l)\rangle = |b\rangle$
4. $|(b \oplus (\tilde{l} \wedge l)) \wedge c\rangle \to |((b \oplus (\tilde{l} \wedge l)) \wedge c) \oplus (b \wedge c)\rangle =: |\tilde{s}\rangle$
5. $|c\rangle \to |c \oplus (\tilde{l} \wedge l)\rangle$

6. $|\tilde{s}\rangle \to |\tilde{s} \oplus (b \wedge (c \oplus (\tilde{l} \wedge l)))\rangle$
7. $|c \oplus (\tilde{l} \wedge l)\rangle \to |c \oplus (\tilde{l} \wedge l) \oplus b\rangle$

Zuerst halten wir fest, dass wir wie gewünscht $|b'\rangle = |b\rangle$ erhalten.
Ist $\tilde{l} = 0$ folgt

$$\begin{aligned} |c'\rangle &= |b \oplus c\rangle, \\ |\tilde{s}\rangle &= |(b \wedge c) \oplus (b \wedge c)\rangle = |0\rangle, \\ |s'\rangle &= |b \wedge c\rangle. \end{aligned}$$

Sei nun $\tilde{l} = 1$. Dann ist

$$\begin{aligned} |c'\rangle &= |b \oplus l \oplus c\rangle, \\ |\tilde{s}\rangle &= |((b \oplus l) \wedge c) \oplus (b \wedge c)\rangle \end{aligned}$$

und daher

$$|\tilde{s}\rangle = \begin{cases} |0\rangle & \text{für } l = 0 \\ |c\rangle & \text{für } l = 1. \end{cases}$$

Weiterhin gilt

$$|s'\rangle = |\tilde{s} \oplus (b \wedge (c \oplus l))\rangle$$

und deshalb

$$|s'\rangle = \begin{cases} |b \wedge c\rangle & \text{für } l = 0 \\ |c \oplus (b \wedge \neg c)\rangle & \text{für } l = 1. \end{cases}$$

Durch Einsetzen der vier möglichen Kombinationen können wir wieder zeigen, dass

$$c \oplus (b \wedge \neg c) = (1 \wedge b) \vee (c \wedge (1 \vee b)).$$

Unser Volladdierer berechnet also das gewünschte Ergebnis.

Den Halbaddierer können wir mittels folgender Schaltung realisieren.

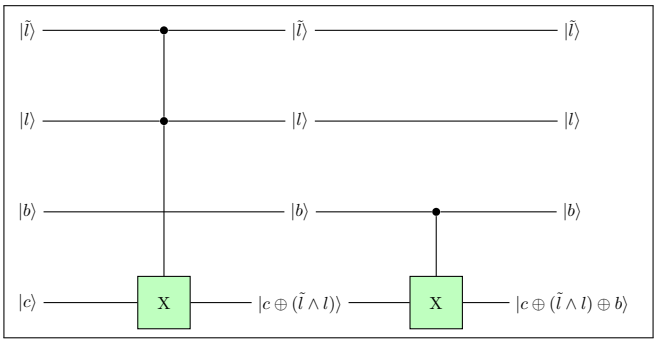

Abbildung 3.6: Halbaddierer im Fall $a = 0, a' = 1$.

- $a = 1, a' = 0$

  Diesen Fall können wir auf den Fall $a = 0, a' = 1$ zurückführen, indem wir zuerst das Kontrollbit l mit einem Pauli-X-Gatter negieren, danach die Schaltungen von Oben anwenden und zum Schluss das Kontrollbit mit einem weiteren Pauli-X-Gatter zurückstellen.

Als nächstes müssen wir ein Verfahren finden, wie wir eine klassische Zahl a mit einer Quantenzahl b vergleichen können. Wir konstruieren diese Operation so, dass wir ein Quantenbit auf $|1\rangle$ setzen, falls $a \geq b$. Dieses können wir als Kontrollbit für unseren Addierer verwenden.
Die Idee unseres Algorithmus besteht darin, dass wir das Bitmuster der Zahlen a und b von links nach rechts lesen, bis sich beide Zahlen an einer Stelle unterscheiden.
Wir erhalten damit den Algorithmus:

Algorithmus 3.3.1. *(Vergleich einer Quantenzahl mit einer klassischen Zahl)*
Eingabe: $a \in \mathbb{N}$, $|R\rangle = |b\rangle|l\rangle|0...0\rangle$ *und* K *die Anzahl der Quantenbits im Register* $|b\rangle$.
Ausgabe: $|b'\rangle|l\rangle|*...*\rangle$ *mit* $l = 1 \Leftrightarrow b < a$.
Sei

$$a := \sum_{i=0}^{K-1} a_i 2^i$$

und

$$b := \sum_{i=0}^{K-1} b_i 2^i$$

mit $a_i, b_i \in \{0, 1\}$.

1. *Für alle* i *von* $K - 1$ *bis* 1*:*
   - *Falls:* $a_i = b_i$*: Setze* $i := i + 1$ *und gehe zu 1.*
   - *Falls:* $a_i = 0$ *und* $b_i = 1$*: Setze* $|l\rangle = |0\rangle$ *und gehe zu 2.*
   - *Falls:* $a_i = 1$ *und* $b_i = 0$*: Setze* $|l\rangle = |1\rangle$ *und gehe zu 2.*
2. *Ende der Schleife.*

Zur Illustration der einzelnen Schritte geben wir auch hier die verwendeten Schaltkreise an:

1. $i = K - 1$: Ist $a_i = 0$ führen wir folgende Operationen aus:

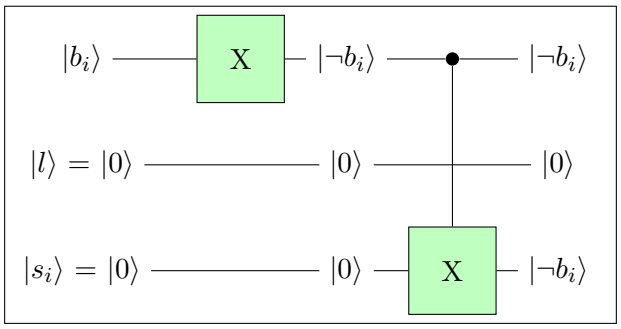

Abbildung 3.7: Vergleich einer Quantenzahl mit einer klassischen Zahl bei $a_{K-1} = 0$.

Mit dieser Schaltung erhalten wir:

$$b_i = 1 : |s_i\rangle \longmapsto |0\rangle \text{ und } |l\rangle \longmapsto |0\rangle,$$
$$b_i = 0 : |s_i\rangle \longmapsto |1\rangle \text{ und } |l\rangle \longmapsto |0\rangle.$$

Ist $a_i = 1$ nutzen wir diesen Schaltkreis:

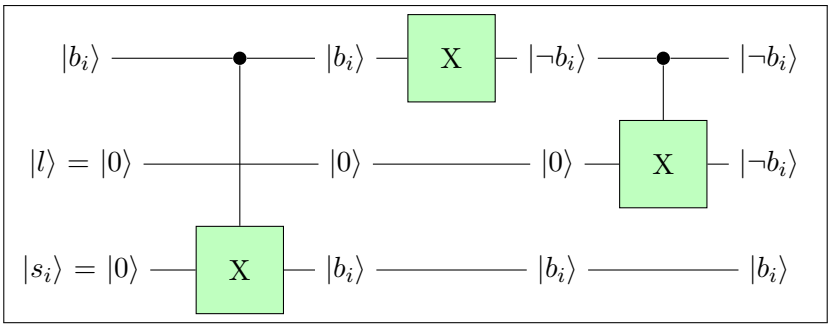

Abbildung 3.8: Vergleich einer Quantenzahl mit einer klassischen Zahl bei $a_{K-1} = 1$.

Mit dieser Schaltung erhalten wir:

$$b_i = 1 : |s_i\rangle \longmapsto |1\rangle \text{ und } |l\rangle \longmapsto |0\rangle,$$
$$b_i = 0 : |s_i\rangle \longmapsto |0\rangle \text{ und } |l\rangle \longmapsto |1\rangle.$$

In beiden Fällen haben wir sichergestellt, dass $|s_i\rangle = |1\rangle$ ist, falls beide Bits gleich sind und $|s_i\rangle = |0\rangle$, falls wir $|l\rangle$ bereits bestimmen konnten.

2. $i \in [1, K-1)$:

   In diesen Schritten verwenden wir jeweils $|s_{i+1}\rangle$ aus der vorangegangenen Stufe als Kontrollbit, da wir den Vergleich nur durchführen müssen, wenn $|l\rangle$ noch nicht bestimmt werden konnte. An einer Stelle werden wir von der in [4] angegebenen Schaltung abweichen, da man an einer Stelle mit CNOT, statt einem Pauli-X-Gatter arbeiten muss.

   Sei $a_i = 0$:

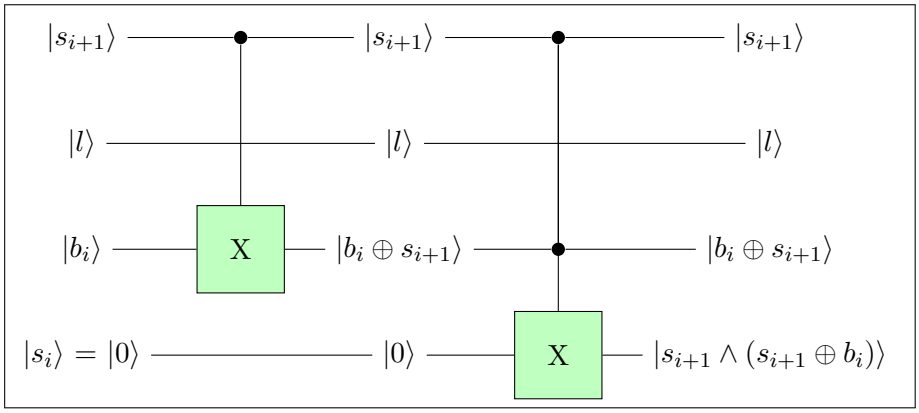

Abbildung 3.9: Vergleich einer Quantenzahl mit einer klassischen Zahl bei $a_i = 0$.

Sei $s_{i+1} = 0$. Wie man leicht sieht, gilt in diesem Fall:

$$b_i \oplus s_{i+1} = b_i \oplus 0 = b_i,$$
$$s_{i+1} \wedge (b_i \oplus s_{i+1}) = 0 \wedge (b_i \oplus s_{i+1}) = 0.$$

Wie gewünscht verändern wir also den Zustand unseres Quantenregisters nicht, da das Kontrollbit nur gesetzt wird, falls der Vergleich im vorherigen Schritt kein Ergebnis geliefert hat.

Ist $s_{i+1} = 1$ erhalten wir:

$$b_i \oplus s_{i+1} = b_i \oplus 1 = \neg b_i,$$
$$s_{i+1} \wedge (b_i \oplus s_{i+1}) = 1 \wedge \neg b_i = \begin{cases} 1 & \text{falls } b_i = 0 \\ 0 & \text{falls } b_i = 1. \end{cases}$$

Da wir sichergehen können, dass das Kontrollbit nur gesetzt ist, falls $|l\rangle = |0\rangle$ erreichen wir in diesem Fall:

$$b_i = 1 : |s_i\rangle \longmapsto |0\rangle \text{ und } |l\rangle \longmapsto |0\rangle,$$
$$b_i = 0 : |s_i\rangle \longmapsto |1\rangle \text{ und } |l\rangle \longmapsto |0\rangle.$$

Sei $a_i = 1$:

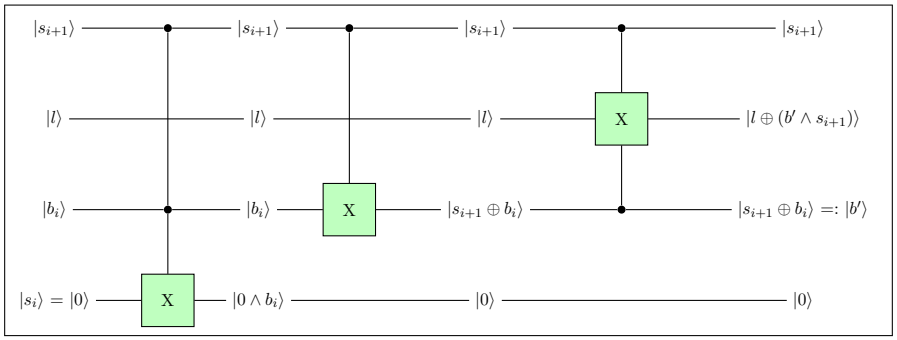

Abbildung 3.10: Vergleich einer Quantenzahl mit einer klassischen Zahl bei $a_i = 1$.

Für $s_{i+1} = 0$ gilt:

$$\begin{aligned} b_i \wedge s_{i+1} &= 0, \\ b_i \oplus s_{i+1} &= b, \\ l \oplus (b' \wedge 0) &= l. \end{aligned}$$

Wir erhalten die Identität.

Sei nun $s_{i+1} = 1$:

$$\begin{aligned} 1 \oplus b_i &= \neg bn \\ 1 \wedge b_i &= \begin{cases} 1 & \text{für } b_i = 1 \\ 0 & \text{für } b_i = 0, \end{cases} \\ l \oplus (\neg b_i \wedge 1) &= \begin{cases} l \oplus 1 = \neg l & \text{falls } b_i = 0 \\ l \oplus 0 = l & \text{falls } b_i = 1. \end{cases} \end{aligned}$$

Da wir nun bereits wissen, dass das Bit $|l\rangle$ ungleich $|1\rangle$ ist, falls das Kontrollbit gesetzt wurde, schließen wir:

$$\begin{aligned} b_i = 1 : &|s_i\rangle \longmapsto |1\rangle \text{ und } |l\rangle \longmapsto |0\rangle, \\ b_i = 0 : &|s_i\rangle \longmapsto |0\rangle \text{ und } |l\rangle \longmapsto |1\rangle. \end{aligned}$$

3. $i = 0$

   Hier müssen wir nur einen Vergleich durchführen, falls $a_0 = 1$ ist, da sonst, falls wir diesen Schritt auswerten müssen, auf alle Fälle $a \leq b$ ist.

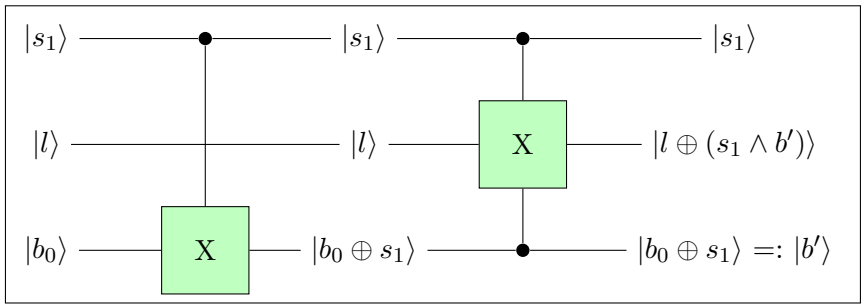

Abbildung 3.11: Vergleich einer Quantenzahl mit einer klassischen Zahl bei $a_0 = 1$.

Auch hier wirken die Operationen nur, falls das Kontrollbit s_1 gesetzt wurde.

Mittels der angegebenen Schaltung können wir testen, ob eine klassische Zahl a größer ist als eine Quantenzahl b und setzen in diesem Fall ein Quantenbit auf 1. Jedoch haben wir um den Vergleich durchzuführen noch weitere Hilfsbits benötigt:

$$LT' : |b\rangle|x\rangle|0...0\rangle \longmapsto |b\rangle|x \oplus y\rangle|g(b))\rangle,$$

wobei

$$y = \begin{cases} 1 & \text{falls } b < a, \\ 0 & \text{falls } b \geq a. \end{cases}$$

und g eine nicht weiter bestimmte Funktion (g für *garbage*). Dass die Hilfsbits nicht notwendigerweise ungleich 0 sind stört uns jedoch, weshalb wir eine Operation der Form

$$LT(a) : |b\rangle|x\rangle|0...0\rangle \longmapsto |b\rangle|x \oplus y\rangle|0...0\rangle$$

konstruieren werden. Dazu wenden wir einen kleinen Trick an:
Die Umkehrabbildung von LT' kann leicht berechnet werden, da sie nur aus selbstinversen Operationen zusammengesetzt ist. Wir müssen den Vergleich daher nur rückwärts ausführen um unerwünschte Zustände zu entfernen. Wenden wir die Umkehrabbildung jedoch sofort an, verlieren wir auch unser „Vergleichsbit" wieder. Verwenden wir noch ein zusätzliches temporäres Bit, können wir das Ergebnis in ein anderes Bit kopieren und erhalten somit:

1. Gegeben ist ein Register $|R\rangle = |b\rangle|x\rangle|0\rangle|0 \ldots 0\rangle$.
2. Berechne $LT'(|b\rangle|0\rangle|0 \ldots 0\rangle) = |b\rangle|y\rangle|g(b))\rangle$ und lasse $|x\rangle$ unverändert.
3. Wende ein kontrolliertes exklusives Oder an:

$$CNOT(|y\rangle|x\rangle) = |y\rangle|x \oplus y\rangle.$$

4. Berechne $LT'^{-1}(|b\rangle|y\rangle|g(b)\rangle) = |b\rangle|0\rangle|0\ldots0\rangle$

Wir benötigen also zur Durchführung des Vergleichs $K+1$ zusätzliche Quantenbits, die am Schluss der Operation wieder auf 0 gesetzt sind und deshalb wieder für andere Teilalgorithmen genutzt werden können.

Bemerkung 3.3.2. *Die temporären Quantenbits sind nach diesem Schritt im Zustand $|0\rangle$. Messen wir diese und erhalten ein anderes Ergebnis liegt ein Fehler im Algorithmus vor und wir können nicht mehr gewährleisten, dass der restliche Algorithmus korrekt abläuft.*
Messen wir jedoch $|0\rangle$ zerstört die Messung alle ,,falschen" Ergebnisse. Mittels dieser Idee könnte man eventuell eine Fehlerkorrektur in Quantenalgorithmen durchführen (vgl. [23] S. 12).

Da wir nun alle benötigten Bausteine besitzen können wir unseren Addierer ADD_a zusammenbauen:

1. Vergleiche mittels der Abbildung LT die klassische Zahl $n-a$ mit der Quantenzahl b. Damit wird $|l\rangle$ auf 1 gesetzt, falls $a+b<n$.

2. $MADD$ addiert nun a, falls $a+b<n$ oder 2^K+a-N falls $a+b\geq n$, wobei im zweiten Fall das letzte Übertragsbit nicht mehr berechnet wird und diese Addition somit äuquivalent zur Addition von $a-n$ ist.

Auf diese Weise kann ein Addierer mit Hilfe von $K+1$ temporären Quantenbits realisiert werden.
Mit einem ähnlichen Trick wie vorhin konstruieren wir ein Verfahren, mit dem wir auch hier den Arbeitsbereich wieder löschen.
Wir nutzen dazu aus, dass $f(b) := a+b \pmod n$ eine umkehrbare Abbildung ist, da $a+b+n-a = b+n = b \pmod n$.
Wir können also eine Quantenoperation konstruieren mit

$$\tilde{F} : |f(b)\rangle|0\rangle \longmapsto |f(b)\rangle|b\rangle.$$

Die Umkehrabbildung hierzu berechnet

$$\tilde{F}^{-1} : |f(b)\rangle|b\rangle \longmapsto |f(b)\rangle|0\rangle.$$

Auch in diesem Fall ist die Umkehrabbildung leicht konstruierbar, indem wir den Algorithmus rückwärts durchlaufen.
Wir können also einen Addierer zusammensetzen, der $|b\rangle$ mit dem Ergebnis von $|a+b \pmod n\rangle$ überschreibt (für weitere Details siehe [4] Kapitel V. D.):

1. Starte mit einem Register $|R\rangle = |b\rangle|0\rangle|0\rangle$

2. Berechne
$$ADD_a(|b\rangle|0\rangle|0\rangle) = |b\rangle|l\rangle|a+b \pmod n\rangle$$
mit $l = 1 \wedge (b + a < n)$.

3. Drehe das Kontrollbit l um:
$$|1 \wedge (a+b<N)\rangle \longmapsto |1 \wedge (a+b \geq N)\rangle.$$

4. Berechne
$$ADD^{-1}_{n-a}(|a+b \pmod n\rangle|\neg l\rangle|b\rangle) = |a+b \pmod n\rangle|0\rangle|0\rangle.$$

Der Addierer benötigt für das Zwischenergebnis K und für die Vergleiche noch einmal $K+1$ temporäre Bits.
Achtung: Wir müssen beim Anwenden von ADD^{-1}_{n-a} die beiden Teilregister vertauschen. Dies können wir entweder klassisch durch Umbenennen der Quantenbits realisieren, oder wie wir auch bei der *Quantenfouriertransformation* sehen werden mit $3K$ CNOT-Gattern.
Mit Hilfe des soeben konstruierten Addierers können wir nun auch einen Multiplizierer konstruieren, indem wir K Addierer hintereinanderschalten.
Um auch hier eine ,,überschreibende" Operation zu konstruieren nutzen wir aus, dass $ggT(a,n) = 1$ und die Multiplikation daher invertierbar ist. Wir können mit Hilfe des erweiterten euklidischen Algorithmus $a^{-1} \pmod n$ effizient auf einem klassischen Rechner berechnen und deshalb mit einem ähnlichen Trick wie im Addierer einen Multiplizierer aus mehreren Addierern zusammensetzen, welcher die Eingabe überschreibt und alle temporären Quantenbits wieder auf $|0\rangle$ setzt. Um diese Operation zu realisieren benötigen wir noch ein weiteres temporäres Bit, weshalb wir mittlerweile bei $2K+2$ temporären Quantenbits angelangt sind.
Mit Hilfe des Multiplizieres können wir Potenzen berechnen (vergleiche Kapitel 3.3.1). Hierbei müssen wir uns aber nicht mehr darum kümmern, dass die Eingabe überschrieben wird, da wir das Ergebnis der Rechnung in einem seperaten Teilregister abspeichern. In diesem Schritt werden keine weiteren temporären Bits benötigt.
Zum Abschluss halten wir noch kurz das Resultat dieses Kapitels fest:

Satz 3.3.3. *Sei K die Anzahl der Quantenbits im Register. Dann können wir mit $\mathcal{O}\left((\log n)^3\right)$ Quantengattern und $\mathcal{O}(\log n)$ zusätzlichen Quantenbits Potenzen berechnen.*

Beweis. Wir haben bereits gezeigt, dass wir $2K+2$ zusätzliche Quantenbits benötigen. Durch Abzählen aller nötigen Operationen kann man zeigen, dass wir mit $\mathcal{O}\left((\log n)^3\right)$ Quantengattern Potenzen auf einem Quantencomputer berechnen können (siehe hierzu auch [4] Kapitel VI B. und die Implementierung des Algorithmus im Begleitprogramm zu dieser Arbeit.). □

3.3.3 Bedeutung für die Praxis

Wir haben nun Möglichkeiten gesehen, wie wir einige Grundrechenarten auf Quantencomputern ausführen können. Eine schöne Eigenschaft daran war die parallele Berechnung der Ergebnisse. Wie wir auch gesehen haben, ist die Anzahl paralleler Berechnungen nur durch die Größe des Quantenregisters begrenzt und geschieht automatisch. Leider erhält man bei einer Messung des Registers nur ein Ergebnis der Berechnung und zerstört gleichzeitig auch die anderen Berechnungen, weshalb diese Algorithmen nicht dazu geeignet sind klassische Verfahren massiv zu beschleunigen. Wie wir jedoch sehen werden ist der Algorithmus von Shor mit einer gewissen Nachbearbeitung in der Lage diese Parallelität gewinnbringend einzusetzen, indem nicht sofort nach der Anwendung der Operation gemessen wird, sondern eine gewisse Nachbereitung stattfindet.

3.4 Die Quantenfouriertransformation

Die *Fouriertransformation* ist ein sehr wichtiges Hilfmittel für Mathematiker, Physiker und Ingenieure. Angewendet wird sie z.B. in der Optik, Akustik, Signalanalyse, Bildbearbeitung und Computeralgebra.
Eine effiziente Möglichkeit zur Berechnung der Fouriertransformation kann somit den Aufwand für eine Vielzahl von Anwendungen senken. Wie wir in Kapitel 4 noch sehen werden, ist auch für den Algorithmus von Shor die Fouriertransformation wichtig.
Ziel dieses Abschnittes ist es, einen Quantenalgorithmus zur Durchführung der Fouriertransformation zu finden. Eine detailliertere Behandlung des Themas findet sich in [16] S.73-82.

3.4.1 Die klassische Fouriertransformation

Bevor wir uns mit der Quantenfouriertransformation beschäftigen, gehen wir kurz auf die *klassische diskrete Fouriertransformation* ein. Diese ist wie folgt definiert:

Definition 3.4.1. *(diskrete Fouriertransformation) Gegeben seien N Daten* $(x_0, ..., x_{N-1})$*, dann ist die* diskrete Fouriertransformation *definiert als die Abbildung*

$$(x_0, ..., x_{N-1}) \to (y_0, ..., y_{N-1}),$$

wobei

$$y_k := \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i \cdot jk/N} x_j.$$

Da jede der N Komponenten des Ergebnisses aus einer Summe mit N Summanden besteht, kann der Aufwand der Fouriertransformation mit $\mathcal{O}(N^2)$

abgeschätzt werden. Benutzen wir $n = \lfloor \log_2 N \rfloor$ Bits zur Darstellung von N, bekommen wir also eine Komplexität von $\mathcal{O}(2^{poly(n)})$. Setzt man die sogenannte Fast Fourier Transformation ein, kann man den Aufwand noch auf Größenordnung $\mathcal{O}(N \log N)$ erniedrigen, verglichen mit der Anzahl der zur Darstellung von N benötigten Bits bleibt der Aufwand jedoch weiterhin exponentiell (vgl. hierzu [7] S.222).

3.4.2 Quantenfouriertransformation

Die Quantenfouriertransformation wird ähnlich zur klassischen Fouriertransformation konstruiert.

Definition 3.4.2. *(Quantenfouriertransformation) Gegeben sei ein Quantenregister mit n Quantenbits. Sei* $|k\rangle \in \{|0\rangle, ..., |N-1\rangle\}$. *Dann ist*

$$QFT|k\rangle := \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i \cdot jk/N} |j\rangle.$$

Wenden wir diese Operation auf ein Quantenregister an, das sich im Zustand $|R\rangle = \sum_{j=0}^{N-1} x_j |j\rangle$ befindet, erhalten wir:

$$\begin{aligned} QFT \sum_{k=0}^{N-1} x_k |k\rangle &= \sum_{k=0}^{N-1} x_j QFT|k\rangle = \\ &= \sum_{k=0}^{N-1} x_k \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i \cdot jk/N} |j\rangle = \\ &= \sum_{j=0}^{N-1} \underbrace{\left(\frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i \cdot jk/N} x_k \right)}_{=:y_j} |j\rangle = \\ &= \sum_{j=0}^{N-1} y_j |j\rangle. \end{aligned}$$

Die Quantenfouriertransformation einer Superposition ist somit gleich der Superposition mit fouriertransformierten Amplituden.
Die Fouriertransformation kann durch folgende Matrix beschrieben werden (Der Exponent wird hierbei als Produkt von Zeilen- und Spaltenindex geschrieben um die Struktur besser sichtbar zu machen):

$$QFT = \frac{1}{\sqrt{N}} \begin{pmatrix} \omega^{0\cdot 0} & \omega^{0\cdot 1} & \dots & \omega^{0\cdot (N-1)} \\ \omega^{1\cdot 0} & \omega^{1\cdot 1} & \dots & \omega^{1\cdot (N-1)} \\ \vdots & \vdots & \vdots & \vdots \\ \omega^{(N-1)\cdot 0} & \omega^{(N-1)\cdot 1} & \dots & \omega^{(N-1)\cdot (N-1)} \end{pmatrix},$$

wobei $\omega := e^{2\pi i/N}$ eine N-te Einheitswurzel ist.

Satz 3.4.3. *Die Quantenfouriertransformation ist eine unitäre Abbildung.*

Beweis. Wir haben bereits gesehen, dass eine Matrix (und daher auch die zugehörige Abbildung) unitär ist, wenn die adjungierte Matrix gleich der Inversen ist. Daher reicht es zu zeigen, dass

$$\frac{1}{N}\sum_{k=0}^{N-1}\omega^{lk}\omega^{-mk} = \delta_{lm}.$$

Fall 1: $l = m$

$$\frac{1}{N}\sum_{k=0}^{N-1}\omega^{lk}\omega^{-lk} = \frac{1}{N}\sum_{k=0}^{N-1}1 = \frac{N}{N} = 1.$$

Fall 2: $l \neq m$

$$\begin{aligned}\frac{1}{N}\sum_{k=0}^{N-1}\omega^{lk}\omega^{-mk} &= \frac{1}{N}\sum_{k=0}^{N-1}\omega^{(l-m)k} = \\ &= \frac{1}{N}\sum_{k=0}^{N-1}e^{2\pi i(l-m)k/N} = \\ &= \frac{1}{N}\sum_{k=0}^{N-1}\underbrace{\left(e^{2\pi i(l-m)/N}\right)^k}_{=:\tilde{\omega}^k}.\end{aligned}$$

Behauptung:

$$\sum_{k=0}^{N-1}\tilde{\omega}^k = 0.$$

Beweis:
Wir nutzen hierbei aus, dass ω eine N-te Einheitswurzel ist, also:

$$\omega^N = 1.$$

Damit folgt sofort, dass auch $\tilde{\omega}$ eine N-te Einheitswurzel ist, denn

$$\tilde{\omega}^N = \left(\omega^{l-m}\right)^N = \left(\omega^N\right)^{l-m} = 1^{l-m} = 1.$$

Obendrein gilt $\tilde{\omega} \neq 1$, da $l \neq m$.
Somit folgt:

$$\begin{aligned}0 = \tilde{\omega}^N - 1 &= (\tilde{\omega}-1)\cdot\left(\tilde{\omega}^{N-1} + \tilde{\omega}^{N-2} + ... + 1\right) = \\ &= \underbrace{(\tilde{\omega}-1)}_{\neq 0}\cdot\sum_{k=0}^{N-1}\tilde{\omega}^k \\ &\Rightarrow \sum_{k=0}^{N-1}\tilde{\omega}^k = 0.\end{aligned}$$

□

3.4.3 Aufwand der Quantenfouriertransformation

Um den Aufwand der Quantenfouriertransformation zu Berechnen, zerlegen wir die Operation in einzelne Quantengatter. Hierfür nutzen wir die Binärdarstellung einer ganzen Zahl ($m_1, ..., m_N \in \{0, 1\}$)

$$m_1 m_2 ... m_N := m_1 \cdot 2^{N-1} + m_2 \cdot 2^{N-2} + ... + j_0 \cdot 2^0 = \sum_{j=0}^{n-1} 2^j \cdot m_{N-j},$$

bzw. für Brüche:

$$0.m_1 m_2 ... m_N := \frac{m_1}{2^1} + \frac{m_2}{2^2} + ... + \frac{m_N}{2^N} = \sum_{j=1}^{n} 2^{-j} \cdot m_j.$$

In dieser Notation gilt

$$m_1 m_2 ... m_N = 2^N \cdot 0.m_1 m_2 ... m_N.$$

Anwenden der Quantenfouriertransformation auf einen Zustand $|j\rangle$ in einem

n-Bit Quantenregister ergibt demnach (mit $N := 2^n$)

$$\begin{aligned}
QFT|j\rangle &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N}|k\rangle \\
&= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk_1...k_n/N}|k\rangle = \\
&= \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi ij0.k_1...k_n}|k\rangle = \\
&= \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi ij\sum_{r=1}^{n} k_r/2^r}|k\rangle = \\
&= \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} \prod_{r=1}^{n} e^{2\pi ijk_r/2^r}|k\rangle = \\
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} ... \sum_{k_n=0}^{1} \left(\prod_{r=1}^{n} e^{2\pi ijk_r/2^r} \right) |k_1...k_n\rangle = \\
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} ... \sum_{k_n=0}^{1} \left(\prod_{r=1}^{n} e^{2\pi ijk_r/2^r} \right) |k_1\rangle...|k_n\rangle = \\
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} ... \sum_{k_n=0}^{1} \left(\prod_{r=1}^{n} e^{2\pi ijk_r/2^r}|k_r\rangle \right) = \\
&= \frac{1}{2^{n/2}} \prod_{r=1}^{n} \sum_{k_r=0}^{1} e^{2\pi ijk_r/2^r}|k_r\rangle = \\
&= \frac{1}{2^{n/2}} \prod_{r=1}^{n} \left(|0\rangle + e^{2\pi ij/2^r}|1\rangle \right).
\end{aligned}$$

In unserer Schreibweise gilt

$$\begin{aligned}
e^{2\pi ij/2^r} &= e^{2\pi i\cdot(j_1j_2...j_{n-r}+0.j_{n-(r-1)}...j_n)} = \\
&= \underbrace{\left(e^{2\pi i}\right)}_{=1}^{j_1j_2...j_{n-r}} \cdot e^{2\pi i\cdot 0.j_{n-(r-1)}...j_n} = \\
&= e^{2\pi i\cdot 0.j_{n-(r-1)}...j_n}.
\end{aligned}$$

Daher erhalten wir

$$QFT|j_1...j_n\rangle = \left(|0\rangle + e^{2\pi i\cdot 0.j_n}|1\rangle\right) ... \left(|0\rangle + e^{2\pi i\cdot 0.j_1...j_n}|1\rangle\right). \tag{3.3}$$

Ein wichtiges Bauteil für die Konstruktion der Quantenfouriertransformation stellt das sogenannte R_k-*Gatter* dar, welches wie folgt definiert ist:

Definition 3.4.4. *Das R_k-Gatter ist ein Quantengatter, das durch folgende Matrix definiert ist:*

$$R_k := \begin{pmatrix} 1 & 0 \\ 0 & \zeta \end{pmatrix}, \text{ mit } \zeta := e^{2\pi i/2^k}$$

Da $\zeta \cdot \bar{\zeta} = 1$ handelt es sich auch hier um eine unitäre Abbildung und unser Gatter ist eine zulässige Quantenoperation. Mit kontrollierten R_k-Gattern und Hadamardgattern können wir uns einen Quantenschaltkreis konstruieren, der die in Formel 3.3 angegebene Abbildung realisiert.

Satz 3.4.5. *Die Quantenfouriertransformation auf n Quantenbits kann mit $\mathcal{O}(n^2)$ Quantengattern realisiert werden.*

Beweis. Wir haben die Quantenfouriertransformation bereits so weit zerlegt, dass wir die elementaren Operationen ablesen können, die wir zur Implementierung benötigen.
Wie bereits in Formel 3.3 gezeigt wurde können wir die Fouriertransformation realisieren, indem wir eine Hadamardtransformation je Quantenbit anwenden (dies erzeugt auch den nötigen Normierungsfaktor) und anschließend noch mittels kontrollierten R_k-Gattern die Amplitude von $|1\rangle$ anpassen.
Man kann nun leicht sehen:

- Beim ersten Quantenbit benötigt man eine Hadamardtransformation.
- Beim zweiten Quantenbit benötigt man eine Hadamardtransformation und ein R_k-Gatter.
- Beim dritten Quantenbit benötigt man eine Hadamardtransformation und zwei R_k-Gatter.
- usw.

Insgesamt brauchen wir also

$$\sum_{i=1}^{n} i = \frac{1}{2} \cdot n \cdot (n+1)$$

Operationen.
Abbildung 3.12 zeigt am Beispiel eines 4-Bit Registers den Aufbau der Quantenfouriertransformation.
Zu beachten ist, dass der hier vorgestellte Schaltkreis die Reichenfolge der Bits vertauscht, was auch im Schaltbild durch die Indizes der Quantenbits angedeutet ist. Wir müssen also nach Anwendung der Fouriertransformation die Quantenbits wieder zurücktauschen. Hierzu gibt es zwei Möglichkeiten:

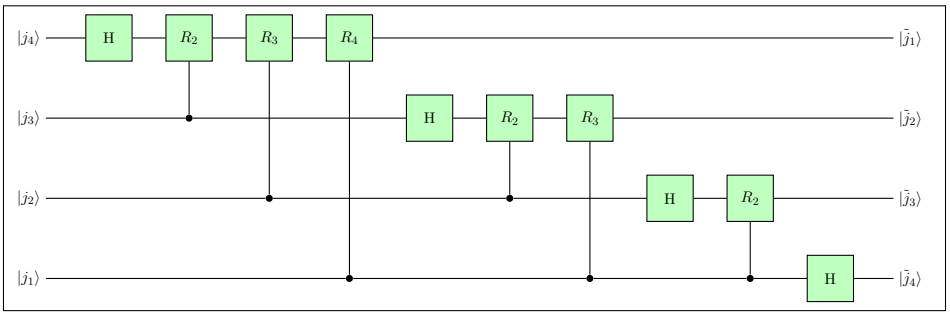

Abbildung 3.12: Quantenfouriertransformation mit 4 Quantenbits, die den Zustand $|j\rangle$ nach $|\tilde{j}\rangle$ überführt.

- Wir konstruieren einen Quantenschaltkreis mit dessen Hilfe wir je zwei Quantenbits vertauschen können (dies kann durch anwenden von je drei CNOT-Gattern erreicht werden, weshalb wir noch $\lfloor \frac{3}{2} \cdot n \rfloor$ Quantengatter zusätzlich benötigen).
- Man vertauscht die Quantenbits rein klassisch, indem man nach der Operation das Quantenregister rückwärts liest.

Der Aufwand die Quantenbits zu vertauschen ist also linear in n, weshalb wir für den Gesamtaufwand der Fouriertransformation $\mathcal{O}(n^2)$ erhalten. □

Bemerkung 3.4.6. *Peter Shor nutzt in seiner Arbeit aus, dass bei den kontrollierten R_k-Gattern das Kontroll- und Zielbit vertauscht werden können ohne etwas an der Wirkung der Operationen zu ändern. Der dort angegebene Schaltkreis weicht also von dem hier verwendeten ab, obwohl er zum gleichen Resultat führt.*

Kapitel 4

Algorithmus von SHOR

Inhaltsangabe

Nachdem wir in den vorherigen Kapiteln gesehen haben, was ein Quantencomputer ist und anhand kleinerer Algorithmen die Vorzüge von Quantenalgorithmen studiert haben, widmen wir uns nun dem eigentlichen Thema der Arbeit, dem *Algorithmus von Shor*.
Problemstellung: Wir haben eine ungerade Zahl $n \in \mathbb{N}$ und wissen, dass es sich hierbei um eine zusammengesetzte Zahl handelt. Wir möchten nun einen Faktor $z \in \mathbb{N}$ finden, so dass $n = z \cdot k$ für ein geeignetes $k \in \mathbb{N}$.
Bevor wir uns diesem Problem zuwenden, führen wir noch wichtige Hilfsmittel ein, die im Algorithmus benötigt werden.

4.1 Der euklidische Algorithmus

Für den Quantenalgorithmus benötigen wir eine zur Eingabe n teilerfremde Zahl, somit müssen wir uns ein Hilfsmittel beschaffen um nachzuprüfen, ob zwei Zahlen teilerfremd sind.
Mit dem *euklidischen Algorithmus* haben wir die Möglichkeit zu ermitteln, wie groß der *größte gemeinsame Teiler* (kurz: *ggT*) zweier ganzer Zahlen ist. Wir definieren:

Definition 4.1.1. *(Teilbarkeit) Seien $a, b \in \mathbb{N}$. Wir nennen a einen* Teiler *von b (Kurzschreibweise $a|b$), wenn es ein $c \in \mathbb{N}$ gibt mit $b = c \cdot a$.*

Definition 4.1.2. *Seien $x, y \in \mathbb{N}$. Die Zahl $z \in \mathbb{N}$ wird definiert als $ggT(x, y)$, wenn gilt:*

1. *z teilt x und z teilt y*
2. *Jedes Element $\tilde{z}$ das x und y teilt, teilt auch z.*

Mit dieser Defninition erhalten wir (vgl. auch [18] S.15):

Satz 4.1.3. *(Euklidischer Algorithmus) Seien $a_0, a_1 \in \mathbb{N}$. Sei a_{i+1} rekursiv definiert durch die Vorschrift*

$$a_{i-1} = q_{i+1}a_i + a_{i+1} \text{ mit } q_{i+1} \in \mathbb{N} \text{ maximal,}$$

solange $a_{k+1} \neq 0$ für ein $k \in \mathbb{N}$.
Dann ist $ggT(a_0, a_1) = a_k$.

Beweis. Um das Resultat zeigen zu können, benötigt man einige kleinere Hilfsaussagen (siehe auch [18] Lemma 3.2).
Seien $a, b, c \in \mathbb{Z}$, dann gilt

1. $ggT(a, 0) = a$,
2. $ggT(a, b) = ggT(b, a)$,
3. $ggT(a, ab) = a$,
4. $ggT(a, b) = ggT(a, b + ca)$.

Damit erhalten wir

$$ggT(a_{i-1}, a_i) = ggT(q_{i+1}a_i + a_{i+1}, a_i) = ggT(a_{i+1}, a_i) = ggT(a_i, a_{i+1})$$

und deshalb ist

$$ggT(a_0, a_1) = ggT(a_1, a_2) = ... = ggT(a_k, \underbrace{a_{k+1}}_{=0}) = a_k.$$

Wir müssen nur noch zeigen, dass es ein $k \in \mathbb{N}$ gibt mit $a_{k+1} = 0$.
Nach unserer Rechenvorschrift gilt für $a_i \neq 0$, dass $a_{i+1} < a_i$. Denn wäre $a_{i+1} \geq a_i$ gäbe es $c, \tilde{c} \in \mathbb{N}$ mit $a_{i+1} = c \cdot a_i + \tilde{c}$ und somit

$$a_{i-1} = q_{i+1}a_i + ca_i + \tilde{c} = (q_{i+1} + c)a_i + \tilde{c}.$$

Dies ist ein Widerspruch zur Maximalität von q_{i+1}.
Da nun zusätzlich $a_i \geq 0$ für alle $i \in \mathbb{N}$ ist, muss ein Index k mit der Eigenschaft $a_k \neq 0$ und $a_{k+1} = 0$ existieren. □

Zur Laufzeitanalyse des Algorithmus formulieren wir die Vorschrift aus Satz 4.1.3 zu folgendem Algorithmus um.

Algorithmus 4.1.4. *(Euklidischer Algorithmus)*
Eingabe: $a, b \in \mathbb{N}$
Ausgabe: *ggT(a, b).*

1. *So lange* $b \neq 0$*:*
2. *Setze* $t := b$
   Setze $b := a \mod b$
   Setze $a := t$
3. *Ende der Schleife.*
4. ***Ausgabe:*** *a.*

Nun bleibt uns nur noch zu zeigen, dass der euklidische Algorithmus effizient arbeitet (siehe hierzu auch [2] Kapitel 2.5.1).

Satz 4.1.5. *Sein* $a, b \in \mathbb{N}$ *mit* $b < a$*, dann hat Algorithmus 4.1.4 eine Laufzeit von* $\mathcal{O}((\log a)^3)$*.*

Beweis. Zuerst betrachten wir das Schleifeninnere:
Am meisten Aufwand bereitet hier die Berechnung von $a \mod b$. Diese Operation kann mit $\mathcal{O}(\log a \cdot \log b)$ berechnet werden (vgl. [2] Proposition 2.4.1). Da aber $b \leq a$ vorausgesetzt wurde können wir das Innere mit Aufwand $\mathcal{O}\left((\log a)^2\right)$ abschätzen.
Nun müssen wir uns noch überlegen, wie oft die Schleife maximal durchlaufen wird, bevor der Algorithmus terminiert.

- Behauptung: Sei f_n die n-te *Fibonacci-Zahl*. Dann wird die Schleife bei Berechnung von $ggT(f_{n+1}, f_n)$ genau $n-1$ mal durchlaufen.

  Beweis:

  Nach Definition der Fibonacci-Zahlen erhalten wir

  $$\begin{aligned} f_{n+1} &= f_n + f_{n-1}, \\ f_n &= f_{n-1} + f_{n-2}, \\ &\vdots \\ f_4 &= f_3 + f_2, \\ f_3 &= f_2 \cdot 2 + 0, \end{aligned}$$

  wobei die letzte Zeile gilt, da $f_1 = f_2 = 1$.

  Wir erhalten also nach $n-1$ Schritten:

  $$ggT(f_{n+1}, f_n) = f_2 = 1$$

  .

- Behauptung: $f_n > \left(\frac{1+\sqrt{5}}{2}\right)^{n-2}$ für $n \geq 3$.

  Beweis: Induktion nach n:

  1. $n = 3$:
  $$f_3 = f_2 + f_1 = 1 + 1 = 2,$$
  $$\frac{1+\sqrt{5}}{2} \approx 1,6 < 2.$$
  2. $n = 4$:
  $$f_4 = f_3 + f_2 = 3,$$
  $$\left(\frac{1+\sqrt{5}}{2}\right)^2 \approx 2,6 < 3.$$
  3. Induktionsschritt:
  $$\begin{aligned} f_n = f_{n-1} + f_{n-2} &> \left(\frac{1+\sqrt{5}}{2}\right)^{n-3} + \left(\frac{1+\sqrt{5}}{2}\right)^{n-4} = \\ &= \left(\frac{1+\sqrt{5}}{2}\right)^{n-4} \cdot \left(1 + \frac{1+\sqrt{5}}{2}\right) = \\ &= \left(\frac{1+\sqrt{5}}{2}\right)^{n-4} \cdot \frac{3+\sqrt{5}}{2} = \\ &= \left(\frac{1+\sqrt{5}}{2}\right)^{n-4} \cdot \frac{6+2\cdot\sqrt{5}}{4} = \\ &= \left(\frac{1+\sqrt{5}}{2}\right)^{n-4} \cdot \frac{1+2\cdot\sqrt{5}+5}{4} = \\ &= \left(\frac{1+\sqrt{5}}{2}\right)^{n-4} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^2 = \\ &= \left(\frac{1+\sqrt{5}}{2}\right)^{n-2}. \end{aligned}$$

- Behauptung: Sei $d := ggT(a, b)$. Bricht der euklidische Algorithmus nach n Schritten ab, dann gilt
  $$a \geq d \cdot f_{n+2},$$
  $$b \geq d \cdot f_{n+1}.$$

  Beweis: Induktion nach n.

1. $n = 1$: Bricht der Algorithmus bereits im ersten Schritt ab ist

$$a = q \cdot b + 0.$$

Wir wissen also, dass b ein Teiler von a ist, und daher gilt

$$ggT(a, b) = b = b \cdot 1 = b \cdot f_2.$$

Obendrein ist

$$a = qb \geq 2 \cdot b = f_3 \cdot b.$$

2. „$n - 1 \to n$”:
Bricht der Algorithmus nach n Schritten ab erhalten wir

$$\begin{aligned} a &= bq_1 + r_1, \\ b &= r_1 q_2 + r_2, \end{aligned}$$

sowie

$$r_i = r_{i+1} q_{i+2} + r_{i+2} \quad \text{für } i \in \{1, ..., n-3\},$$

und

$$r_{n-2} = r_{n-1} q_n + 0.$$

Wir wissen, dass

$$d := ggT(b, r_1) = ggT(a, b)$$

und erhalten nach Induktionsannahme

$$\begin{aligned} b &\geq d \cdot f_{n+1}, \\ r_1 &\geq d \cdot f_n. \end{aligned}$$

Damit folgt

$$a = r_1 + bq_1 \geq r_1 + b \geq d \cdot (f_{n+1} + f_n) = d \cdot f_{n+2}$$

und unsere Behauptung ist bewiesen.

- Behauptung: Die Anzahl der Schleifendurchläufe ist $\mathcal{O}(\log b)$.

Beweis: Wie wir in den vorangegangenen Schritten schon gesehen haben können wir die Anzahl der benötigten Schritte mit Hilfe der Fibonaccizahlen abschätzen.

So benötigen wir maximal n Schritte, falls $a > b \geq f_{n+1}$. Wie wir bereits gezeigt haben ist

$$b \geq f_{n+1} > \left(\frac{1 + \sqrt{5}}{2} \right)^{n-1}$$

und man kann leicht nachrechnen, dass

$$\log_{10} \frac{1+\sqrt{5}}{2} \approx 0,209 > \frac{1}{5}.$$

Daraus folgern wir

$$\log_{10} b > \log_{10} \left(\left(\frac{1+\sqrt{5}}{2} \right)^{n-1} \right) > \frac{n-1}{5}$$

und

$$n < 1 + 5 \log_{10} b = \mathcal{O}(\log b).$$

Wir haben gesehen, dass wir $\mathcal{O}(\log b)$ Scheifendurchläufe benötigen und jeder Durchlauf die Komplexität $\mathcal{O}\left((\log a)^2\right)$ hat. Da $a \geq b$ ist $\log a \geq \log b$. Die Gesamtlaufzeit des Algorithmus kann daher mit $\mathcal{O}\left((\log a)^3\right)$ abgeschätzt werden. □

Korollar 4.1.6. *Seien $a, b \in \mathbb{N}$ und $g := ggT(a, b)$. Dann existieren $x, y \in \mathbb{Z}$ mit*

$$a \cdot x + b \cdot y = g$$

Beweis. Um diese Aussage zu beweisen müssen wir den euklidischen Algorithmus erweitern, dass ein passendes Paar (x, y) berechnet wird (siehe hierzu [2] 1.3.3 und [18] Satz 3.9). □

4.2 Kettenbruchapproximation

Ein weiteres wichtiges Hilfsmittel, das wir bereitstellen müssen, ist ein Verfahren um *Näherungsbrüche* bestimmen zu können. Hierzu bietet sich die *Kettenbruchapproximation* an (vgl. [18] Kapitel 10). Unter einem *Kettenbruch* (engl.: *continued fraction*) versteht man einen Ausdruck der Form

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \ldots}}}.$$

In unserem Fall können wir uns auf $a_0 \in \mathbb{Z}$ und $a_i \in \mathbb{N}$ für alle $i \in \mathbb{N}$ beschränken.
Da diese Notation sehr unübersichtlich werden kann führen wir eine Kurzschreibweise für Kettenbrüche ein.

Definition 4.2.1. *Seien $a_0 \in \mathbb{Z}$ und $a_i \in \mathbb{N}$ für alle $i \in \mathbb{N}$. Dann ist:*

$$[a_0, a_1, a_2, a_3, \ldots] := a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \ldots}}}$$

Bevor wir uns damit beschäftigen, wie wir mit diesem Hilfsmittel Näherungsbrüche berechnen können, betrachten wir zunächst ein Verfahren, wie man eine reelle Zahl in einen Kettenbruch umwandeln kann.

Algorithmus 4.2.2. *Der Kettenbruchalgorithmus*
Eingabe: $x \in \mathbb{R}$
Ausgabe: $x' = [a_0, a_1, a_2, \ldots]$ *mit* $x = x'$

1. *Setze* $a_0 := \lfloor x \rfloor$ *und* $t_0 := x - a_0$
2. *Solange* $t_n \neq 0$*:*
   - *Setze* $\xi_n := \frac{1}{t_n}$
   - *Setze* $a_{n+1} = \lfloor \xi_n \rfloor$
   - *Setze* $t_{n+1} = \xi_n - a_{n+1}$

Satz 4.2.3. *Algorithmus 4.2.2 wandelt eine reelle Zahl in den zugehörigen Kettenbruch um.*

Beweis. Um diesen Satz zu beweisen weichen wir leicht von unserer Notation ab und erlauben in der letzten Komponente des Kettenbruchs eine reelle Zahl. Wir müssen damit nur noch zeigen, dass:

$$x = [a_0, a_1, \ldots, a_n, \xi_n] \text{ für alle } n \in \mathbb{N},$$

wobei $a_0 \in \mathbb{Z}$, $a_i \in \mathbb{N}$ für alle $i = 1, \ldots, n$ und $\xi_n \in \mathbb{R}$. Durch Induktion nach n erhalten wir:

- $n = 0$:
$$x = [x] = [a_0 + t_0] = \left[a_0 + \frac{1}{\xi_0}\right] = [a_0, \xi_0].$$
- „$n \to n + 1$": Nach Induktionsvoraussetzung ist
$$x = [a_0, \ldots, a_n, \xi_n].$$
Weiterhin gilt
$$a_{n+1} + t_{n+1} = a_{n+1} + \xi_n - a_{n+1} = \xi_n,$$
womit wir folgern können, dass
$$\begin{aligned} x &= [a_0, \ldots, a_n, a_{n+1} + t_{n+1}] = \\ &= \left[a_0, \ldots, a_n, a_{n+1} + \frac{1}{\xi_{n+1}}\right] = \\ &= [a_0, \ldots, a_n, a_{n+1}, \xi_{n+1}]. \end{aligned}$$

Hiermit haben wir gezeigt, dass wir beim Durchführen eines Schritts den Wert des Kettenbruchs nicht ändern. □

Wir sehen in diesem Algorithmus, dass das Abbruchkriterium nicht immer erfüllt sein muss, weswegen wir ein Kriterium angeben, welches uns garantiert, dass der Algorithmus terminiert und wir ihn somit für unsere Zwecke einsetzen können.

Lemma 4.2.4. *Algorithmus 4.2.2 liefert genau dann einen endlichen Kettenbruch* $[a_0, a_1, \ldots, a_n]$ *mit* $a_0 \in \mathbb{Z}, a_i \in \mathbb{N}$ *für* $i = 1, \ldots, n$*, wenn* $x \in \mathbb{Q}$*.*

Beweis. (Vergleiche [18] S. 68/69)

- „⇒":

  Der Kettenbruchalgorithmus bricht ab, wenn $t_n = 0$ für ein $n \in \mathbb{N}$. Nach Satz 4.2.3 wissen wir, dass

  $$\begin{aligned} x &= [a_0, \ldots, a_{n-1}, \xi_{n-1}] = \\ &= [a_0, \ldots, a_{n-1}, a_n + t_n] = \\ &= [a_0, \ldots, a_{n-1}, a_n]. \end{aligned}$$

  Bricht der Algorithmus ab, erhalten wir einen endlichen Kettenbruch. Damit muss $x \in \mathbb{Q}$ gelten.

- „⇐":

  Sei nun $x \in \mathbb{Q}$. Wir müssen in diesem Fall zeigen, dass unser Algorithmus abbricht.

  Hierbei nutzen wir aus, dass der Kettenbruchalgorithmus nur eine Umformulierung des euklidischen Algorithmus darstellt. Da wir schon gezeigt haben, dass der euklidische Algorithmus nur endlich viele Schritte $n \in \mathbb{N}$ benötigt, bricht der Kettenbruchalgorithmus ebenfalls nach endlich vielen Schritten ab.

  Sei also $x := \frac{p}{q}$ mit $p, q \in \mathbb{Z}$. Wir definieren nun mit Hilfe des euklidischen Algorithmus

  $$r_0 := q,$$

  $$p = {a_0}' \cdot r_0 + r_1$$

  mit $0 < r_1 < r_0$,

  $$r_{i-2} = {a_{i-1}}' \cdot r_{i-1} + r_i$$

  mit $0 < r_i < r_{i-1}$ und $i = 2, \ldots, n-1$ und

  $$r_{n-1} = {a_n}' r_n + 0,$$

womit wir erhalten:

$$x = \frac{p}{q} = \frac{p}{r_0} = {a_0}' + \frac{r_1}{r_0} = {a_0}' + \frac{1}{\frac{r_0}{r_1}}$$

mit $\frac{1}{\frac{r_0}{r_1}} \in [0, 1)$, sowie

$$\frac{r_{i-1}}{r_i} = {a_i}' + \frac{1}{\frac{r_i}{r_{i+1}}}$$

mit $\frac{1}{\frac{r_i}{r_{i+1}}} \in [0, 1)$ für alle $i = 1, \ldots, n-1$ und

$$\frac{r_{n-1}}{r_n} = {a_n}'.$$

Definieren wir nun $a_i := {a_i}'$ für $i = 1, \ldots, n$ und $\xi_i := \frac{r_i}{r_{i+1}}$ für $i = 1, \ldots, n-1$ erhalten wir die Rechenvorschrift aus unserem Kettenbruchalgorithmus.

□

Wir haben gerade gesehen, dass wir mit Hilfe des Kettenbruchalgorithmus eine rationale Zahl in einen endlichen Kettenbruch umwandeln können. Die Kettenbruchdarstellung einer Zahl können wir nun gewinnbringend einsetzen, indem wir uns ein Verfahren konstruieren mit dessen Hilfe wir Näherungsbrüche berechnen. Wir betrachten hierzu ein Verfahren, welches eigentlich zur Annäherung von Zahlen $x \in \mathbb{R} \setminus \mathbb{Q}$, wie z.B. e oder π, genutzt wird. Wir werden das Verfahren jedoch auf rationale Zahlen anwenden.
Hierzu definieren wir (vgl. [18] S. 70/71):

$$p_{-2} := 0,\ p_{-1} := 1,\ q_{-2} := 1,\ q_{-1} := 0,$$

und

$$\begin{aligned} p_i &:= a_i \cdot p_{i-1} + p_{i-2} \quad \text{für } i \in \mathbb{N}, \\ q_i &:= a_i \cdot q_{i-1} + q_{i-2} \quad \text{für } i \in \mathbb{N}. \end{aligned}$$

Man kann nun zeigen (siehe [18] S. 72, Satz 10.9), dass für $i \geq 1$

$$\left| x - \frac{p_i}{q_i} \right| < \frac{1}{q_i q_{i+1}} \leq \frac{1}{i \cdot (i+1)} \tag{4.1}$$

ist.
Die Idee hinter unserem Algorithmus ist, dass wir eine gegebene Zahl x in einen Kettenbruch umwandeln und gleichzeitig $\frac{p_i}{q_i}$ berechnen, bis der Abstand $\left| x - \frac{p_i}{q_i} \right|$ kleiner als eine vorgegebene Schranke ist. Die Ungleichung (4.1) stellt hierbei sicher, dass die Näherungsbrüche gegen das exakte Ergebnis konvergieren.

Algorithmus 4.2.5. *(Kettenbruchapproximation)*
Eingabe: $x \in \mathbb{R}$, $\varepsilon > 0$
Ausgabe: $p, q \in \mathbb{Z}$ *mit* $\left|x - \frac{p}{q}\right| < \varepsilon$

1. *Setze* $\xi := x$, $p_0 := 0$, $q_0 := 1$, $p_1 := 1$ *und* $q_1 := 0$.
2. *Berechne*
$$\tilde{p} := a \cdot p_1 + p_0$$
und
$$\tilde{q} := a \cdot q_1 + q_0$$
mit $a := \lfloor \xi \rfloor$. *Setze anschließend* $t := \xi - a$, $p_0 := p_1$, $p_1 := \tilde{p}$, $q_0 := q_1$, $q_1 := \tilde{q}$ *und* $\xi := \frac{1}{t}$
3. *Falls* $\left|x - \frac{p_1}{q_1}\right| \geq \varepsilon$ *gehe zu 2.*
4. *Ausgabe:* p_1, q_1

Satz 4.2.6. *Sei* $x = \frac{p}{q}$ *und* $a := \max\{p, q\}$. *Dann kann die Kettenbruchapproximation mit Aufwand* $\mathcal{O}\left((\log a)^3\right)$ *durchgeführt werden.*

Beweis. Wie wir im Beweis zu Lemma 4.2.4 bereits gesehen haben ist der Kettenbruchalgorithmus eng verwandt mit dem euklidischen Algorithmus. Daher können wir auch hier zeigen (siehe Beweis zu Satz 4.1.5), dass wir $\mathcal{O}(\log a)$ Schleifendurchläufe benötigen. Im Inneren der Schleife müssen wir zwei Multiplikationen und eine Division (je $\mathcal{O}\left((\log a)^2\right)$) und zusätzlich noch eine Subtraktion und zwei Additionen (je $\mathcal{O}\left(\log a\right)$) ausführen.
Der Aufwand im Inneren ist somit

$$3\mathcal{O}\left((\log a)^2\right) + 3\mathcal{O}(\log a) = \mathcal{O}\left((\log a)^2\right).$$

□

4.3 Algorithmus von Shor

Bisher wurde noch kein effizientes Verfahren zur *Faktorisierung natürlicher Zahlen* mittels klassischen Computern gefunden. Der Algorithmus von Shor profitiert jedoch von den Vorteilen eines Quantencomputers und kann mit polynomialen Aufwand Faktoren einer Zahl finden.
Ziel dieses Abschnitts ist es den Algorithmus zu konstruieren und zu analysieren. Das Kapitel hält sich größtenteils an [23], [21] und [13]. Da es sich nicht um einen reinen Quantenalgorithmus handelt, werden wir zuerst den klassischen Teil und im Anschluss daran den Quantenalgorithmus behandeln.

4.3.1 Klassischer Teil

Definition 4.3.1. *Sei $n \in \mathbb{N}$. Wir definieren*

- $\mathbb{Z}_n := \mathbb{Z}/n\mathbb{Z}$,
- $\mathbb{Z}_n^* := \{z \in \mathbb{Z}_n : z^{-1} \in \mathbb{Z}_n\}$, *die* Einheitengruppe *von* $\mathbb{Z}_n$,
- $ord_n(x) := r \in \mathbb{N}$ *mit* $x^r = 1 \pmod{n}$ *und* $x^k \neq 1 \pmod{n}$ *für alle* $0 < k < r$. *Wir nennen* r *in diesem Fall die* Ordnung *von* x.

Wir werden ein Kriterium herleiten, wie wir feststellen können, ob eine Zahl in der Einheitengruppe $\mathbb{Z}_n^*$ liegt. Dafür müssen wir kurz einen Blick zu den sogenannten *linearen diophantischen Gleichungen* werfen (vgl. [2] Abschnitt 1.3.4 und [18] Satz 5.10).

Satz 4.3.2. *Seien $a, b, c \in \mathbb{Z}$ mit $a, b \neq 0$.*
Dann gibt es genau dann $x, y \in \mathbb{Z}$ mit

$$ax + by = c,$$

wenn

$$ggT(a, b)|c.$$

Beweis. Siehe hierzu auch [2] Proposition 1.3.11.

- Seien $x, y \in \mathbb{Z}$ mit $ax + by = c$ und $g := ggT(a, b)$.
  Da $g|a$ und $g|b$ gilt auch $g|(ax + by)$ und somit $g|c$
- Sei $g := ggT(a, b)$. Gilt nun $g|c$, dann finden wir ein $h \in \mathbb{Z}$ mit $c = g \cdot h$.
  Mit Hilfe von Korollar 4.1.6 finden wir $\tilde{x}, \tilde{y} \in \mathbb{Z}$ mit

  $$g = a\tilde{x} + b\tilde{y}$$

  Multipliziert man diese Gleichung nun mit h erhalten wir

  $$ah\tilde{x} + bh\tilde{y} = hg = c.$$

  Nun müssen wir nur noch $x := h\tilde{x}$ und $y := h\tilde{y}$ setzen.

□

Satz 4.3.3. *Sei $n \in \mathbb{N}$. Dann ist*

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n : ggT(a, n) = 1\}.$$

Beweis. Sei $a \in \mathbb{Z}_n^*$. Dann existiert ein $x \in \mathbb{Z}_n$ mit

$$a \cdot x = 1 \pmod{n}.$$

Das heißt, es gibt ein $y \in \mathbb{Z}$ mit

$$a \cdot x = 1 + y \cdot n \Leftrightarrow a \cdot x - y \cdot n = 1.$$

Wie wir bereits in Satz 4.3.2 gesehen haben ist diese Gleichung genau dann lösbar, wenn $ggT(a,n)|1$. Daraus folgt aber sofort $ggT(a,n) = 1$. □

Shor's Algorithmus baut auf der Idee auf eine Zahl n zu faktorisieren indem man die Ordnung eines Elements $x \in \mathbb{Z}_n^*$ bestimmt.
Die zentrale Idee hierzu liefert der folgende Satz:

Satz 4.3.4. *Sei $n \in \mathbb{N}$ und $x \in \mathbb{Z}_n^*$ mit $ord_n(x) = r$. Dann liefert*

$$ggT(x^{\frac{r}{2}} \pm 1, n)$$

einen nicht trivialen Teiler von n, falls r gerade und

$$x^{\frac{r}{2}} \neq -1 \pmod{n}.$$

Beweis. Es gilt

$$ord_n(x) = r,$$

weswegen

$$x^r = 1 \pmod{n}$$

und

$$x^r - 1 = 0 \pmod{n}.$$

Da r gerade ist, erhalten wir

$$\left(x^{\frac{r}{2}} - 1\right) \cdot \left(x^{\frac{r}{2}} + 1\right) = 0 \pmod{n}$$

und

$$\left(x^{\frac{r}{2}} - 1\right) \cdot \left(x^{\frac{r}{2}} + 1\right) = k \cdot n,$$

wobei $k \in \mathbb{Z}$.

- Behauptung: n teilt nicht $x^{\frac{r}{2}} - 1$

  Annahme: n teilt $x^{\frac{r}{2}} - 1$. Dann ist

  $$x^{\frac{r}{2}} - 1 = 0 \pmod{n}$$

  und daher

  $$x^{\frac{r}{2}} = 1 \pmod{n},$$

  was einen Widerspruch zur Minimalität von r darstellt.

- Behauptung: n teilt nicht $x^{\frac{r}{2}}+1$

  Annahme: n teilt $x^{\frac{r}{2}}+1$. Dies ist genau dann der Fall, wenn

  $$x^{\frac{r}{2}} = -1 \pmod{n},$$

  und wir haben einen Widerspruch zur Voraussetzung.

Damit ist

$$\boxed{ggT(x^{\frac{r}{2}} \pm 1, n) \neq n.}$$

Es folgt auch sofort, dass $ggT(x^{\frac{r}{2}} \pm 1, n) \neq 1$, denn wäre $ggT(x^{\frac{r}{2}}+1, n) = 1$ dann müsste n ein Teiler von $x^{\frac{r}{2}}-1$ sein. Analog müsste $n| \left(x^{\frac{r}{2}}-1\right)$ gelten, falls $ggT(x^{\frac{r}{2}}-1, n) = 1$. □

Wir wählen also eine zufällige Zahl x aus der Menge $\{2, ..., n-1\}$. Danach bestimmen wir den größten gemeinsamen Teiler von x und n um sicherzustellen, dass $x \in \mathbb{Z}_n^*$. Ist $ggT(x, n) \neq 1$ haben wir einen Teiler von n gefunden und können abbrechen. Sind jedoch x und n teilerfremd können wir einen Teiler ermitteln indem wir $ord_n(x)$ bestimmen und überprüfen, ob die Bedingungen aus Satz 4.3.4 erfüllt sind. Abbildung Abbildung 4.1 zeigt den Ablauf des Algorithmus auf.
Wie wir später sehen werden liefert der Quantenalgorithmus nur zu einer gewissen Wahrscheinlichkeit die Ordnung zurück. Wir haben also eine gewisse Unsicherheit im Algorithmus. Um die Wahrscheinlichkeit zu erhöhen, einen Teiler zu finden, stellen wir noch gewisse Anforderungen an die zu zerlegende Zahl n, welche wir jedoch mit polynomiellen Aufwand überprüfen können.
Dies wären:

- n darf keine Primzahl sein, da sonst der Algorithmus von Shor nicht terminiert (in polynomieller Zeit nachprüfbar z.B. deterministisch mittels AKS-Test [1]).
- n darf keine Primzahlpotenz sein (in polynomieller Zeit nachprüfbar, siehe [2] S. 288, Proposition 6.3.7).

Ein weiteres Problem des Algorithmus ist, dass man nur einen Teiler ermitteln kann, wenn die gefundene Ordnung gerade ist. Findet man heraus, dass die Elementordnung ungerade ist, muss man den Algorithmus abbrechen. Mit folgendem Satz können wir die Fehlerquote des Algorithmus abschätzen.

Satz 4.3.5. *Sei $n \in \mathbb{N}$ eine ungerade und aus mindestens 2 verschiedenen Primzahlen zusammengesetzte Zahl. Sei weiterhin $x \in [2, n-1]$ zufällig gewählt. Dann beträgt die Wahrscheinlichkeit einen Teiler zu finden mindestens $1 - \frac{1}{2^{k-1}}$, wobei k die Anzahl der verschiedenen ungeraden Primteiler von n bezeichnet.*

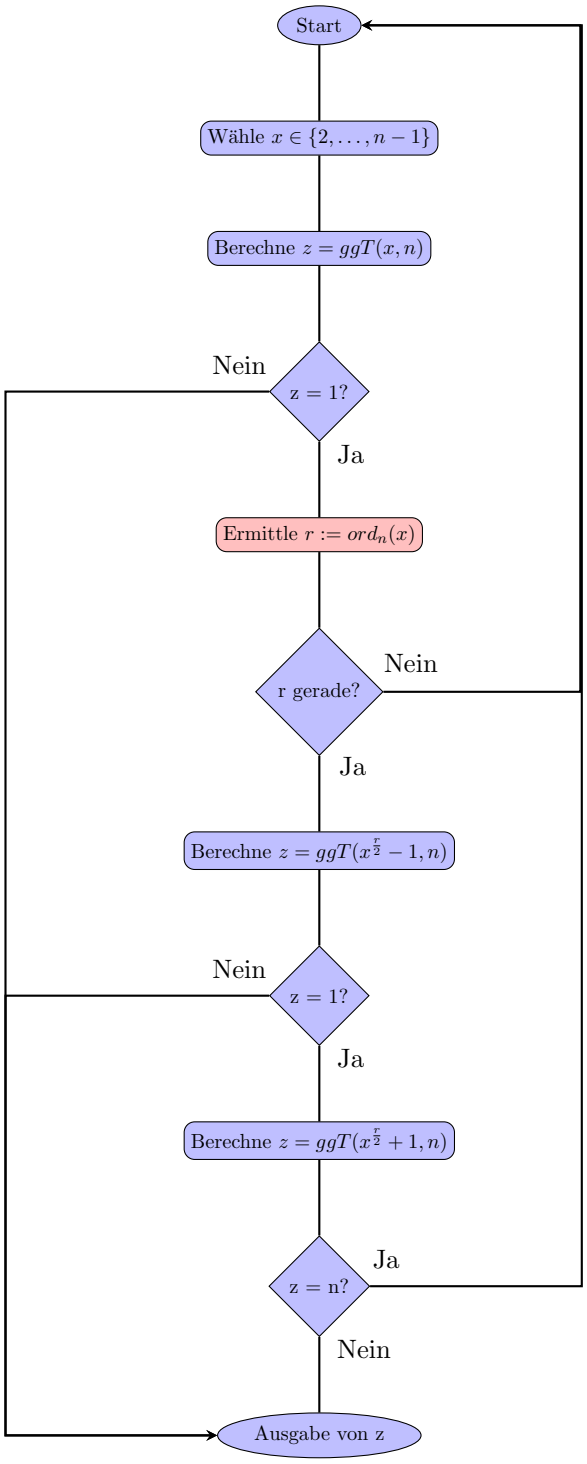

Abbildung 4.1: Flußdiagramm zum Algorithmus von Shor.

Die Erfolgswahrscheinlichkeit des Algorithmus kann also mit mindestens 50% abgeschätzt werden.

Zum Beweis dieser Aussage benötigen wir noch einige Hilfsmittel.

Lemma 4.3.6. *(Chinesischer Restsatz)*
Seien $m, n \in \mathbb{N}$ teilerfremd. Dann ist die Abbildung

$$\begin{aligned} \Phi : \mathbb{Z}_{nm} &\longrightarrow \mathbb{Z}_n \times \mathbb{Z}_m, \\ x + nm\mathbb{Z} &\longmapsto (x + n\mathbb{Z}, x + m\mathbb{Z}) \end{aligned}$$

ein Isomorphismus.

Beweis. Siehe [18] Satz 5.5. □

Lemma 4.3.7. *Sei $n = p^\alpha$ mit p ungerade Primzahl und $\alpha \in \mathbb{N}$. Dann ist $\mathbb{Z}_n^*$ eine zyklische Gruppe, d.h. es gibt ein $g \in \mathbb{Z}_n$ mit $\mathbb{Z}_n^* = \{g^0, g^1, g^2, \ldots, g^{r-1}\}$, wobei $r = \varphi(n)$.*

Beweis. Siehe [18] Satz 7.7. □

Lemma 4.3.8. *Sei $n \in \mathbb{N}$ eine Primzahlpotenz. Dann gibt es genau ein Element der Ordnung 2 in $\mathbb{Z}_n^*$.*

Beweis. Sei $x = 1$, dann ist $ord_n(x) = 1 < 2$.
Sei also $x \neq 1$ aus $\mathbb{Z}_n^*$ mit $x^2 = 1$. Somit gilt

$$x^2 - 1 = 0 \pmod{n},$$

genau dann, wenn

$$(x - 1) \cdot (x + 1) = 0 \pmod{n}.$$

Da $\mathbb{Z}_n^*$ eine zyklische Gruppe ist, ist diese auch nullteilerfrei, d.h.

$$a \cdot b = 0 \pmod{n}$$

genau dann, wenn

$$a = 0 \pmod{n} \text{ oder } b = 0 \pmod{n}.$$

Wir wissen, dass

$$x \neq 1 \pmod{n}$$

und daher ist

$$x - 1 \neq 0 \pmod{n}.$$

Der zweite Faktor muss also 0 sein und deshalb ist

$$x + 1 = 0 \pmod{n},$$

was äquivalent ist zu

$$x = -1 \pmod{n}$$

□

Beweis von Satz 4.3.5. Wir betrachten die Primfaktorzerlegung von n. Da n ungerade und keine Primzahlpotenz ist, gilt

$$n = \prod_{i=1}^{k} {p_i}^{\alpha_i} \text{ mit } p_i \text{ ungerade.}$$

Sei $r_i = ord_{{p_i}^{\alpha_i}}(x)$. Die Ordnung von $x \pmod n$ ist dann

$$r = kgV(r_1, ..., r_k).$$

Sei nun $q_i \in \mathbb{N}_0$ die größte Zahl, so dass $2^{q_i} | r_i$ für alle $i = 1, ..., k$.

1. $q_i = 0$ für alle $i = 1, ..., k$:

   In diesem Fall sind alle r_i ungerade, weshalb auch r ungerade ist. Der Algorithmus liefert kein Ergebnis.

2. $q_i = q > 0$ für alle $i = 1, ..., k$:

   Hier gilt (mit Lemma 4.3.8)

   $$x^{\frac{r}{2}} = -1 \pmod{p_i^{\alpha_i}}$$

   und anwenden des chinesischen Restsatzes liefert

   $$x^{\frac{r}{2}} = -1 \pmod{p_1^{\alpha_1} \cdot ... \cdot p_k^{\alpha_k}} = -1 \pmod n.$$

   Somit erhalten wir auch in diesem Fall keine Lösung.

3. $q_i \neq q_j$ für mindestens ein $i \neq j$: In diesem Fall funktioniert unser Algorithmus, da einerseits r gerade ist und mindestens ein Index μ existiert mit

   $$x^{\frac{r}{2}} \neq -1 \pmod{p_\mu^{\alpha_\mu}}.$$

   Nach dem chinesischem Restsatz ist somit auch

   $$x^{\frac{r}{2}} \neq -1 \pmod n.$$

Ein zufälliges $x \in \mathbb{Z}_n^*$ zu ziehen ist äquivalent dazu, dass man für alle $i \in \{1, \ldots, k\}$ ein $x_i \in \mathbb{Z}_{{p_i}^{\alpha_i}}$ zieht. Nach Lemma 4.3.7 existiert ein Erzeuger g_i mit $x_i = g_i^{s_i} \pmod{p_i^{\alpha_i}}$.
Wir wissen bereits, dass

$$ord_{p_i^{\alpha_i}}(g_i) = \varphi\left(p_i^{\alpha_i}\right) = p_i^{\alpha_i - 1}(p_i - 1) =: 2^{m_i} o_i \text{ mit } o_i \text{ ungerade.}$$

Siehe hierzu auch [18] Lemma 5.14.

Man kann nun zeigen, dass die Wahrscheinlichkeit $P(q_i = j) \leq \frac{1}{2}$ für $i \in \{1, ..., k\}$ und $j \in \{0, ..., m_i\}$ (siehe [9] Anhang B).
Die Wahrscheinlichkeit, dass alle q_i gleich sind, ist folglich

$$p \leq \prod_{i=1}^{k-1} \frac{1}{2} = \frac{1}{2^{k-1}}.$$

□

Wie wir nun gesehen haben ist die Wahrscheinlichkeit keinen Teiler zu finden immer $\leq 50\%$. Durch Einführung eines weiteren Tests könnte man die Fehlerwahrscheinlichkeit noch auf $\leq 25\%$ erniedrigen (siehe hierzu auch [17]).

Satz 4.3.9. *Sei $n = p \cdot q$ mit p, q zwei verschiedene, ungerade Primzahlen. Sei $x \in \mathbb{Z}_n^*$ zufällig gewählt und $\left(\frac{x}{n}\right) = -1$, wobei $\left(\frac{x}{n}\right)$ das sogenannte* Jacobi Symbol *ist (vgl. [18] Definition 8.10).*
Dann ist die Wahrscheinlichkeit mindestens $\frac{3}{4}$, dass $ord_n(x) =: r$ gerade ist und $x^{r/2} \neq -1 \pmod{n}$.

Beweis. Siehe [17] Theorem 4. □

4.3.2 Quantenmechanischer Teil

In diesem Abschnitt widmen wir uns nun dem Teil des Algorithmus, der ihn effizient werden lässt. Die Grundidee hierbei ist sehr ähnlich zum Algorithmus von Deutsch (Siehe Algorithmus 3.2.2) und profitiert wie dieser von der gleichzeitigen Auswertung einer Funktion an mehreren Stellen.
Zur Bestimmung der Ordnung eines zufällig gewählten $x \in \mathbb{Z}_n^*$ arbeiten wir mit einem Quantenregister der Länge L, wobei L so gewählt wird, dass $n^2 \leq 2^L < 2 \cdot n^2$ und einem zusätzlichen Register der Länge $\frac{L}{2}$. Wie wir in Abschnitt 3.3 gesehen haben benötigen wir obendrein noch $L + 2$ temporäre Quantenbits, die jedoch die meiste Zeit im Zustand $|0\rangle$ sind. Wir geben daher im Folgenden nur noch die beiden Arbeitsregister an.
Der Quantenteil des Algorithmus besteht somit aus den folgenden Schritten (siehe: [23] S. 16/17 und [13] S. 224 ff.):

Algorithmus 4.3.10. *(Quantenalgorithmus zur Bestimmung der Ordnung)*
Eingabe: *$n \in \mathbb{Z}$, $x \in \{2, ..., n-1\}$ mit $ggT(x, n) = 1$*

1. *$R = |a\rangle|b\rangle \leftarrow |0 \ldots 0\rangle|0 \ldots 0\rangle$*
2. *Hadamard Transformation auf das Register $|a\rangle$ anwenden.*
3. *Berechne $|b\rangle = x^{|a\rangle} \pmod{n}$ mittels der in Kapitel 3.3 vorgestellten Methode.*
4. *Messen von $|b\rangle$.*

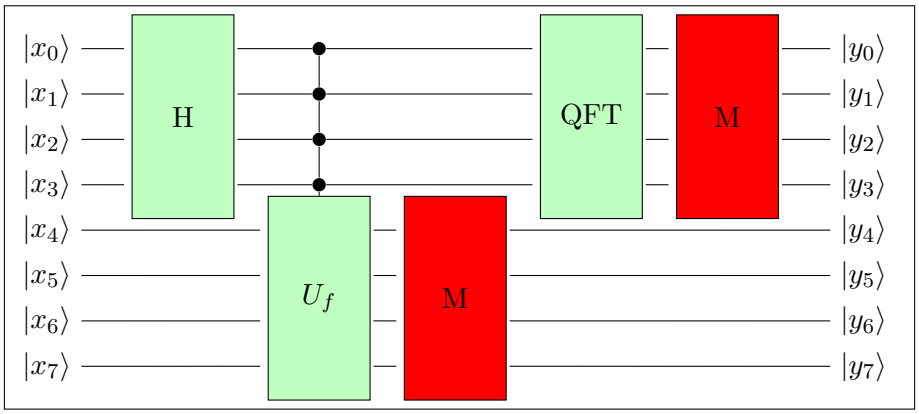

Abbildung 4.2: Schaltkreis zum Quantenalgorithmus mit 8 Quantenbits. Das erste Register wurde hier kleiner als nötig gewählt, damit der Schaltkreis übersichtlicher bleibt.

5. *Anwenden der Quantenfouriertransformation auf* $|a\rangle$.
6. *Messen von* $|a\rangle$.
7. *Berechne aus dem Messwert die Ordnung* r *von* $x \pmod n$ *und gib das Ergebnis aus, falls* r *gerade ist und* $x^{\frac{r}{2}} \neq -1 \pmod n$.

Wir definieren $q := 2^L$. In Schritt 2 des Algorithmus bringen wir unser Quantenregister in die Form

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle|0\rangle.$$

Schritt 3 versetzt das Register dann in den Zustand

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle|x^a \pmod n\rangle.$$

Die weitere Analyse des Algorithmus teilen wir auf und behandeln zuerst einen etwas einfacheren Fall.

1. q ist ein Vielfaches von r

   Durch die Messung des zweiten Registers in Schritt 4 erhalten wir einen Wert $x^k \pmod n$. Da durch die Operation in Schritt 3 die beiden Register verschränkt wurden beeinflusst sie auch das Register $|a\rangle$ mit und wir erhalten eine Superposition aller möglichen Urbilder unseres Messergebnisses:

$$|R\rangle = \frac{1}{\sqrt{\hat{q}}} \cdot \sum_{j=0}^{\hat{q}-1} |k + jr \pmod q\rangle|x^k \pmod n\rangle,$$

wobei $\hat{q} = \frac{q}{r}$.

Die Fouriertransformation in Schritt 5 bewirkt

$$|k + jr\rangle \longmapsto \frac{1}{\sqrt{q}} \sum_{y=0}^{q-1} \omega^{(k+jr)y} |y\rangle,$$

mit $\omega := \exp\left(\frac{2\pi i}{q}\right)$.

Wir erhalten nach Anwendung der Fouriertransformation

$$\begin{aligned} |R\rangle &= \frac{1}{\sqrt{\hat{q}q}} \sum_{j=0}^{\hat{q}-1} \sum_{y=0}^{q-1} \omega^{(k+jr)y} |y\rangle |x^k \pmod n\rangle = \\ &= \frac{1}{\sqrt{\hat{q}q}} \sum_{y=0}^{q-1} \omega^{ky} \sum_{j=0}^{\hat{q}-1} \omega^{jry} |y\rangle |x^k \pmod n\rangle. \end{aligned}$$

Die Wahrscheinlichkeit y zu messen ist

$$P(y) = \frac{1}{\hat{q}q} \cdot \underbrace{\left|\omega^{ky}\right|^2}_{=1} \cdot \left| \sum_{j=0}^{\hat{q}-1} \omega^{jry} \right|^2 = \frac{1}{\hat{q}q} \cdot \left| \sum_{j=0}^{\hat{q}-1} \omega^{jry} \right|^2.$$

Da nach unserer Annahme q ein Vielfaches von r ist, folgern wir

$$\omega^{jry} = \exp\left(\frac{2\pi ijry}{q}\right) = \exp\left(\frac{2\pi ijy}{\hat{q}}\right) =: \gamma^{jy}$$

und somit

$$P(y) = \frac{1}{\hat{q}q} \left| \sum_{j=0}^{\hat{q}-1} \gamma^{jy} \right|^2.$$

- Sei $y = l\hat{q} < q$ mit $l \in \mathbb{N}_0$. Dann gilt

  $$\gamma^{jy} = \exp\left(\frac{2\pi ijy}{\hat{q}}\right) = \exp\left(2\pi ijl\right) = 1$$

  und die Wahrscheinlichkeit y zu messen ist

  $$P(y) = \frac{1}{\hat{q}q} \left| \sum_{j=0}^{\hat{q}-1} 1 \right|^2 = \frac{\hat{q}}{q} = \frac{1}{r}.$$

- Sei $0 < y < q$ kein Vielfaches von $\hat{q}$. Dann ist

  $$P(y) = 0$$

  (Siehe hierzu auch Beweis von Satz 3.4.3.)

Die Messung in Schritt 6 liefert uns demnach mit Sicherheit ein ganzzahliges Vielfaches von $\hat{q} = \frac{q}{r}$.

Beispiel 4.3.11. *Zur Illustration betrachten wir, wie der Quantenalgorithmus funktioniert, wenn wir $x = 7$ und $n = 15$ wählen. Wir werden jedoch im ersten Register nur 4 Quantenbits verwenden um den Zustand etwas übersichtlicher angeben zu können. Somit ist $q = 2^4 = 16$ (vgl. [13] Beispiel 8.6).*

(a) Nach der Hadamardtransformation erhalten wir ein Register der Form

$$\frac{1}{4} \cdot \sum_{a=0}^{15} |a\rangle|0\rangle.$$

(b) Wir berechnen jetzt $x^a \pmod n$ und bringen das Register in den Zustand

$$\frac{1}{4} \cdot \sum_{a=0}^{15} |a\rangle|x^a \pmod n\rangle.$$

(c) Wie man leicht nachrechnen kann ist $ord_{15}(7) = 4$. Damit erhalten wir nach der Messung des zweiten Registers einen der Zustände

$$\begin{aligned}
|R\rangle_1 &= \frac{1}{2} \cdot (|0\rangle + |4\rangle + |8\rangle + |12\rangle) \cdot |1\rangle, \\
|R\rangle_2 &= \frac{1}{2} \cdot (|1\rangle + |5\rangle + |9\rangle + |13\rangle) \cdot |7\rangle, \\
|R\rangle_3 &= \frac{1}{2} \cdot (|2\rangle + |6\rangle + |10\rangle + |14\rangle) \cdot |4\rangle, \\
|R\rangle_4 &= \frac{1}{2} \cdot (|3\rangle + |7\rangle + |11\rangle + |15\rangle) \cdot |13\rangle.
\end{aligned}$$

(d) Wenden wir nun die Fouriertransformation auf das erste Teilregister an erhalten wir

$$\begin{aligned}
QFT(|R\rangle_1) &= \frac{1}{2} \cdot (|0\rangle + |4\rangle + |8\rangle + |12\rangle) \cdot |1\rangle, \\
QFT(|R\rangle_2) &= \frac{1}{2} \cdot (|0\rangle + i|4\rangle - |8\rangle - i|12\rangle) \cdot |7\rangle, \\
QFT(|R\rangle_3) &= \frac{1}{2} \cdot (|0\rangle - |4\rangle + |8\rangle - |12\rangle) \cdot |4\rangle, \\
QFT(|R\rangle_4) &= \frac{1}{2} \cdot (|0\rangle - i|4\rangle - |8\rangle + i|12\rangle) \cdot |13\rangle.
\end{aligned}$$

Dies gilt, da in unserem Fall

$$\omega = \exp\left(\frac{2\pi}{16}i\right)$$

ist und

$$P(y) = 0 \text{ für } y \neq \{0, 4, 8, 12\}.$$

Obendrein wissen wir, dass

$$QFT(|R\rangle) = \frac{1}{\sqrt{4 \cdot 16}} \sum_{y=0}^{15} \omega^{ky} \sum_{j=0}^{3} \omega^{jry} |y\rangle |x^k \pmod n\rangle,$$

sowie

$$\begin{aligned} \omega^{\alpha} &= \exp\left(\frac{2\pi i}{16} \cdot \alpha\right) = \\ &= \exp\left(\frac{2\pi i}{16} \cdot (\alpha \mod 16)\right) = \\ &= \omega^{(\alpha \mod 16)}. \end{aligned}$$

Da $ry = 4 \cdot l \cdot 4 = l \cdot 16$ *gilt*

$$\sum_{j=0}^{3} \omega^{jry} |y\rangle = 4|y\rangle,$$

und

$$QFT(|R\rangle) = \frac{1}{2}\left(|0\rangle + \omega^{4k}|4\rangle + \omega^{8k}|8\rangle + \omega^{12k}|12\rangle\right)|x^k \pmod n\rangle$$

Tabelle 4.1 gibt die Koeffizienten der einzelnen Zustände an und Abbildung 4.3 zeigt die Wahrscheinlichkeitsverteilung der Messwerte, nach Anwenden der Fouriertransformation.

k	$\omega^{(0k \mod 16)}$	$\omega^{(4k \mod 16)}$	$\omega^{(8k \mod 16)}$	$\omega^{(12k \mod 16)}$
0	$\omega^0 = 1$	$\omega^0 = 1$	$\omega^0 = 1$	$\omega^0 = 1$
1	$\omega^0 = 1$	$\omega^4 = i$	$\omega^8 = -1$	$\omega^{12} = -i$
2	$\omega^0 = 1$	$\omega^8 = -1$	$\omega^0 = 1$	$\omega^8 = -1$
3	$\omega^0 = 1$	$\omega^{12} = -i$	$\omega^8 = -1$	$\omega^4 = i$

Tabelle 4.1: Koeffizienten nach der Fouriertransformation in Abhängigkeit von k.

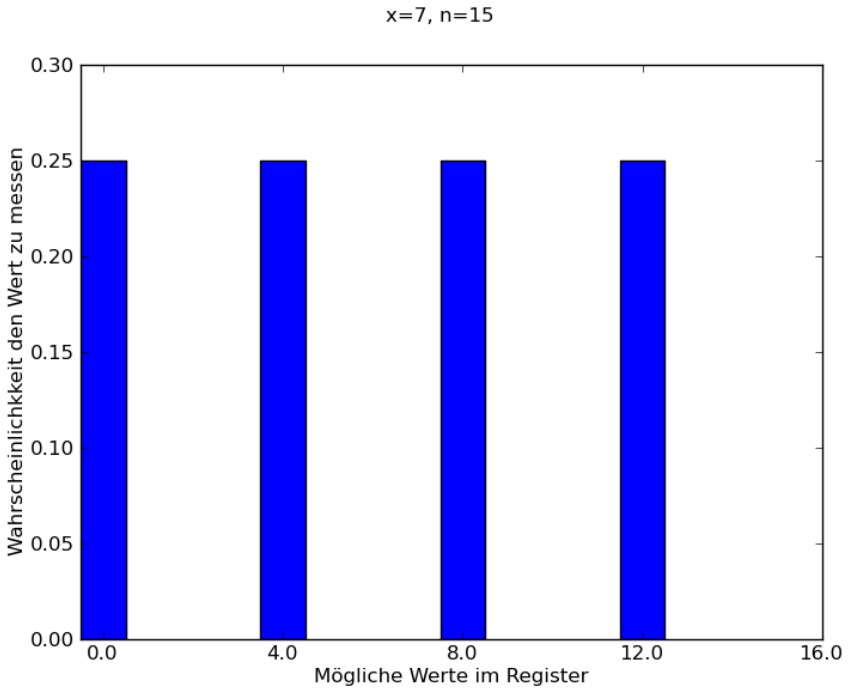

Abbildung 4.3: Visualisierung der Wahrscheinlichkeitsverteilung der Messwerte nach Anwenden der Fouriertransformation. Gewählt wurde $q = 16$. Die Ordnung teilt in diesem Fall q weshalb wir nur diese vier Werte messen können.

2. q ist kein Vielfaches von r

   Auch in diesem Fall befindet sich das Register nach der Messung im Zustand

$$|R\rangle = \frac{1}{\sqrt{\hat{q}}} \cdot \sum_{j=0}^{\hat{q}-1} |k + jr \pmod q\rangle |x^k \pmod n\rangle,$$

   jedoch können wir hier nicht mehr angeben, wie groß $\hat{q}$ genau ist. Es lässt sich allerdings zeigen, dass $\hat{q} \in \left\{ \left\lfloor \frac{q}{r} \right\rfloor, \left\lceil \frac{q}{r} \right\rceil \right\}$ (siehe [13] S.229).

   Wie im vorher behandelten Fall befindet sich das Register nach Anwenden der Fouriertransformation im Zustand

$$|R\rangle = \frac{1}{\sqrt{\hat{q}q}} \sum_{y=0}^{q-1} \omega^{ky} \sum_{j=0}^{\hat{q}-1} \omega^{jry} |y\rangle |x^k \pmod n\rangle$$

   und die Wahrscheinlichkeit einen Wert y zu messen beträgt

$$P(y) = \frac{1}{\hat{q}q} \cdot \left| \sum_{j=0}^{\hat{q}-1} \omega^{jry} \right|^2 .$$

   Ab hier unterscheiden sich die beiden Fälle. Da wir im vorherigen Fall genau wussten wie groß $\hat{q}$ ist konnten wir die Wahrscheinlichkeit explizit berechnen. Wie wir jedoch gleich noch sehen werden kann diese Summe abgeschätzt werden.

   Zuerst stellen wir fest, dass es sich um eine Partialsumme der geometrischen Reihe handelt, und wir daher

$$\sum_{j=0}^{\hat{q}-1} \omega^{jry} = \sum_{j=0}^{\hat{q}-1} (\omega^{ry})^j = \frac{\omega^{\hat{q}ry} - 1}{\omega^{ry} - 1},$$

   schreiben können, wobei auch hier

$$\omega := \exp\left(\frac{2\pi i}{q}\right).$$

   Es gilt

$$\begin{aligned} P(y) &= \frac{1}{\hat{q}q} \left| \frac{\omega^{\hat{q}ry} - 1}{\omega^{ry} - 1} \right|^2 = \\ &= \frac{1}{\hat{q}q} \cdot \frac{\left(\exp\left(\frac{2\pi i}{q}\hat{q}ry\right) - 1\right)\left(\exp\left(-\frac{2\pi i}{q}\hat{q}ry\right) - 1\right)}{\left(\exp\left(\frac{2\pi i}{q}ry\right) - 1\right)\left(\exp\left(-\frac{2\pi i}{q}ry\right) - 1\right)}. \end{aligned}$$

Wir nutzen aus, dass

$$\begin{aligned}(\exp(i\alpha)-1)\,(\exp(-i\alpha)-1) &= 1-\exp(i\alpha)-\exp(-i\alpha)+1 = \\ &= 2-(\exp(i\alpha)+\exp(-i\alpha)) = \\ &= 2-2\cos(\alpha)\end{aligned}$$

und erhalten

$$P(y) = \frac{1}{\hat{q}q}\cdot\frac{2-2\cos\left(2\pi\frac{ry\hat{q}}{q}\right)}{2-2\cos\left(2\pi\frac{ry}{q}\right)}.$$

Ist $\frac{ry}{q}\in\mathbb{Z}$, dann ist dieser Bruch unbestimmt, da Zähler und Nenner gleich 0 sind. Wir können trotzdem den Wert dieses Bruchs angeben. Dazu setzen wir im Folgenden

$$z := \frac{2\pi ry}{q}.$$

Mittels Taylorentwicklung (siehe [5] S. 433) in $z = z_0$ erhalten wir

$$\begin{aligned}\cos(\hat{q}\cdot z) = \cos(\hat{q}\cdot z_0) &- \sin(\hat{q}\cdot z_0)\cdot\hat{q}\cdot(z-z_0) - \\ &- \frac{1}{2}\cos(\hat{q}\cdot z_0)\cdot\hat{q}^2\cdot(z-z_0)^2 + \mathcal{O}((z-z_0)^3)\end{aligned}$$

und

$$\begin{aligned}\cos(z) = \cos(z_0) &- \sin(z_0)\cdot(z-z_0) - \\ &- \frac{1}{2}\cos(z_0)\cdot(z-z_0)^2 + \mathcal{O}((z-z_0)^3).\end{aligned}$$

Ist $z_0 = m\cdot 2\cdot\pi$, mit $m\in\mathbb{Z}$, dann gilt

$$\begin{aligned}\sin(\hat{q}\cdot z_0) &= 0,\\ \sin(z_0) &= 0,\\ \cos(\hat{q}\cdot z_0) &= 1,\\ \cos(z_0) &= 1\end{aligned}$$

und deshalb

$$\lim_{z\to z_0}\frac{1-\cos(\hat{q}z)}{1-\cos(z)} = \lim_{z\to z_0}\frac{\frac{1}{2}\hat{q}^2(z-z_0)^2+\mathcal{O}((z-z_0)^3)}{\frac{1}{2}(z-z_0)^2+\mathcal{O}((z-z_0)^3)} = \hat{q}^2.$$

Wir erhalten als Wahrscheinlichkeit diesen Wert zu messen daher

$$P(y) = \frac{\hat{q}^2}{\hat{q}q} = \frac{\hat{q}}{q} = \frac{1}{r}-\frac{1}{q} \text{ weil } \frac{q}{r}-1\le\hat{q}\le\frac{q}{r}+1.$$

Da q mit steigendem n wächst und r nicht direkt von n abhängt, können wir folgern (vgl. auch [23] S. 17/18, [9] Abschnitt VII. und [20] S. 300), dass

$$P(y) \approx \frac{1}{r} \text{ für ein ausreichend großes } n.$$

Sei nun $\frac{ry}{q} \notin \mathbb{Z}$. Dann ist $\cos\left(2\pi\frac{ry}{q}\right) \neq 1$ und der Nenner $\neq 0$. Betrachten wir deshalb, was mit dem Zähler des Bruchs geschehen kann:

- $ry\hat{q}$ ist ein ganzzahliges Vielfaches von q. Unter dieser Voraussetzung ist $\frac{ry\hat{q}}{q} = \alpha \in \mathbb{Z}$ und $\cos(2\pi\alpha) = 1$. Daher ist

$$P(y) = 0.$$

- $ry\hat{q}$ ist kein ganzzahlig Vielfaches von q. Dann ist

$$P(y) = \frac{1}{\hat{q}q} \cdot \frac{1 - \cos\left(2\pi\frac{ry\hat{q}}{q}\right)}{1 - \cos\left(2\pi\frac{ry}{q}\right)}.$$

  Sei $z := ry \mod q$, dann gibt es ein $k \in \mathbb{Z}$ mit $ry + kq = z$ und wir erhalten

$$\begin{aligned}\cos\left(2\pi\frac{z\hat{q}}{q}\right) &= \cos\left(2\pi\frac{(ry+kq)\hat{q}}{q}\right) = \\ &= \cos\left(2\pi k\hat{q} + 2\pi\frac{ry\hat{q}}{q}\right) = \\ &= \cos\left(2\pi\frac{ry\hat{q}}{q}\right).\end{aligned}$$

  Die Wahrscheinlichkeit dafür ein y in der Nähe eines ganzzahligen Vielfachen von $\frac{q}{r}$ zu messen können wir wie folgt abschätzen.
  Sei $|ry \mod q| \leq \frac{r}{2}$, dann gibt es ein $d \in \mathbb{Z}$ mit der Eigenschaft

$$|ry - dq| \leq \frac{r}{2} \Leftrightarrow |y - d\frac{q}{r}| \leq \frac{1}{2}.$$

  Wir erhalten

$$\begin{aligned}\frac{2\pi ry}{q} &\in \left[\frac{2\pi r}{q}\left(\frac{dq}{r} - \frac{1}{2}\right), \frac{2\pi r}{q}\left(\frac{dq}{r} + \frac{1}{2}\right)\right] = \\ &= \left[2\pi d - \frac{\pi r}{q}, 2\pi d + \frac{\pi r}{q}\right]\end{aligned}$$

und

$$\frac{2\pi r y \hat{q}}{q} \in \left[\frac{2\pi r\hat{q}}{q}\left(\frac{dq}{r}-\frac{1}{2}\right), \frac{2\pi r\hat{q}}{q}\left(\frac{dq}{r}+\frac{1}{2}\right)\right] = \\ = \left[2\pi d\hat{q} - \frac{\pi r\hat{q}}{q}, 2\pi d\hat{q} + \frac{\pi r\hat{q}}{q}\right].$$

Da der Kosinus eine Periode von 2π hat, können wir die ganzzahligen Vielfachen von 2π in den obigen Intervallen ignorieren. Wir definieren

$$\theta := \frac{2\pi r y}{q}, \text{ mit } |\theta| \leq \frac{\pi r}{q}.$$

Daraus folgt

$$\begin{aligned} P(y) &= \frac{1}{\hat{q}q} \cdot \frac{1-\cos(\hat{q}\theta)}{1-\cos(\theta)} \overset{\text{Taylorentwicklung}}{=} \\ &= \frac{1}{\hat{q}q} \cdot \frac{\theta^2\hat{q}^2 - \frac{1}{6}\theta^3\hat{q}^3\sin(\xi)}{\theta^2 - \frac{1}{6}\theta^3\sin(\kappa)} = \\ &= \frac{\hat{q}}{q} \cdot \frac{1-\frac{1}{6}\theta\hat{q}\sin(\xi)}{1-\frac{1}{6}\theta\sin(\kappa)} \geq \\ &\geq \frac{\hat{q}}{q} \cdot \frac{1-\frac{1}{6}\theta\hat{q}\sin(\xi)}{1+\frac{1}{6}\frac{\pi r}{q}} \geq \\ &\geq \frac{\hat{q}}{q} \cdot \frac{1-\frac{1}{6}\frac{\pi\hat{q}r}{q}}{1+\frac{1}{6}\frac{\pi r}{q}} = \\ &= \frac{\hat{q}}{q} \cdot \frac{6q-\pi\hat{q}r}{6q+\pi r} = \\ &= \hat{q} \cdot \frac{6-\frac{\pi\hat{q}r}{q}}{6q+\pi r}. \end{aligned}$$

Nun nutzen wir wieder aus, dass

$$\frac{1}{r} - \frac{1}{q} \leq \frac{\hat{q}}{q} \leq \frac{1}{r} + \frac{1}{q},$$

und folgern

$$\begin{aligned} P(y) &\geq \hat{q} \cdot \frac{6 - \pi r \left(\frac{1}{r} + \frac{1}{q}\right)}{6q + \pi r} = \\ &= \hat{q} \cdot \frac{6-\pi}{6q+\pi r} - \underbrace{\frac{\hat{q}}{q} \cdot \frac{\pi r}{6q+\pi r}}_{=:R_1} \geq \\ &\geq \left(\frac{q}{r} - 1\right) \cdot \frac{6-\pi}{6q+pir} - R_1 = \\ &= \underbrace{\frac{q}{r} \cdot \frac{6-\pi}{6q+\pi r}}_{=:P} - \underbrace{\frac{6-\pi}{6q+\pi r}}_{=:R_2} - R_1 \end{aligned}$$

Betrachten wir diese drei Terme seperat, können wir R_1, R_2 und P abschätzen:

$$\begin{aligned} R_1 &= \frac{\hat{q}}{q} \cdot \frac{\pi r}{6q+\pi r} \overset{\hat{q}<q}{<} \frac{\pi r}{6q+\pi r} < \frac{\pi}{6} \cdot \frac{r}{q} \\ R_2 &= \frac{6-\pi}{6q+\pi r} \leq \frac{6-\pi}{6} \cdot \frac{1}{q} \\ P &= \frac{q}{r} \cdot \frac{6-\pi}{6r + \pi\frac{r^2}{q}} \geq \frac{6-\pi}{6r^2+\pi r^2} = \\ &= \frac{6-\pi}{(6+\pi)r^2} \geq \frac{2}{10r^2} = \frac{1}{5r^2} \end{aligned}$$

Satz 4.3.12. *Sei $n \in \mathbb{N}$ die mit Hilfe des Algorithmus zu zerlegende Zahl und q, r wie oben definiert. Sei $y \in \mathbb{Z}$ ein möglicher Messwert des Quantenalgorithmus, mit $|ry \mod q| \leq \frac{r}{2}$. Dann ist*

$$P(y) \geq \frac{1}{5r^2} + \mathcal{O}\left(\frac{1}{n}\right).$$

Gilt zusätzlich noch $n \geq 10r^2$, so erhalten wir

$$P(y) \geq \frac{1}{10r^2}.$$

Beweis. Wie wir bereits gesehen haben ist

$$P(y) \geq \frac{1}{5r^2} - R_1 - R_2,$$

wobei R_1, R_2 wie oben definiert sind. Wir müssen somit nur noch

zeigen, dass $R_1 + R_2 \leq \frac{1}{10r^2}$.

$$\begin{aligned} R_1 + R_2 &\leq \frac{\pi}{6} \cdot \frac{r}{q} + \frac{6-\pi}{6} \cdot \frac{1}{q} = \\ &= \left(\frac{\pi r - \pi}{6} + 1 \right) \frac{1}{q} = \\ &= \left(\frac{\pi}{6}(r-1) + 1 \right) \frac{1}{q} \leq \\ &\leq \frac{r+1}{q} \leq \frac{n}{n^2} = \frac{1}{n} \leq \frac{1}{10r^2}, \end{aligned}$$

wobei die letzte Abschätzung mit der Voraussetzung $n \geq 10r^2$ gilt. □

Bemerkung 4.3.13. *Wählen wir $n \in \mathbb{N}$ groß genug, können wir den Term $\mathcal{O}\left(\frac{1}{n}\right)$ vernachlässigen und die Wahrscheinlichkeit mit $P(y) \geq \frac{1}{5r^2}$ abschätzen. In der Literatur zum Algorithmus von Shor finden sich noch weitere Strategien zur Schätzung dieser Wahrscheinlichkeit.*

Peter Shor selbst verzichtet auf die Messung vor der Quantenfouriertransformation und schätzt die Wahrscheinlichkeit einen Wert nahe $\frac{q}{r}$ zu messen ab, indem er die Summe mit einem Integral abschätzt (siehe [23] S. 16-18).

Er kann dadurch ermitteln, dass

$$P(y) \geq \frac{1}{3r^2} + \mathcal{O}\left(\frac{1}{q}\right) \approx \frac{1}{3r^2}.$$

John Preskill arbeitet in seinem Beweis (siehe [20] S. 300) auch mit der geometrischen Reihe und erhält die Abschätzung

$$P(y) \geq \frac{4}{\pi^2} \frac{1}{r}.$$

Auch Artur Ekert und Richard Jozsa erhalten mit einer ähnlichen Herangehensweise in ihrem Beweis diese Grenze (siehe [9] Abschnitt VII.).

Beispiel 4.3.14. *Wie wir leicht nachrechnen können erhalten wir den hier analysierten Fall, wenn wir $ord_{77}(13)$ bestimmen wollen, da $10 = ord_{77}(13)$ keine Zweierpotenz ist und somit nicht $q = 2^L$ teilt. Führen wir alle Schritte im Quantenalgorithmus durch, erhalten wir die in Abbildung 4.4 und 4.5 angegebene Wahrscheinlichkeitsverteilung. Man kann gut erkennen, dass die Verteilung in der Nähe ganzzahliger Vielfacher von $\frac{q}{r}$ Peaks hat und somit die Wahrscheinlichkeit einen derartigen Wert zu messen relativ hoch ist. Angegeben ist ein*

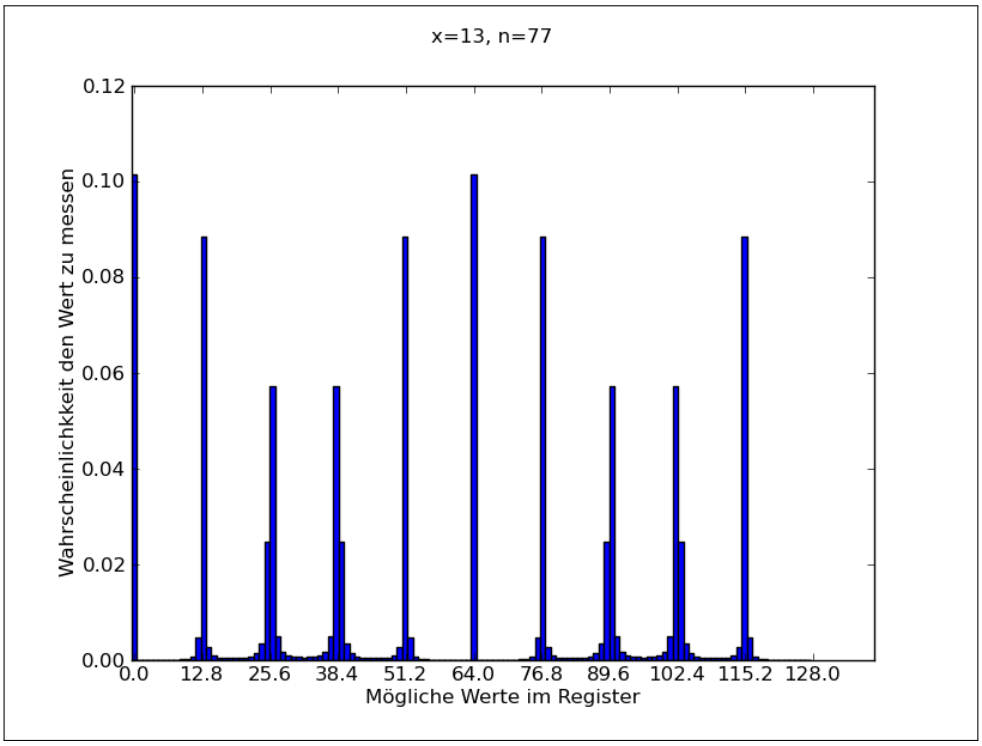

Abbildung 4.4: Wahrscheinlichkeit gewisse Werte nach Anwenden der Fouriertransformation zu messen. Gewählt wurde $q = 128$ (d.h. Anzahl der Bits $< \log_2 77^2$) damit man die Struktur der Wahrscheinlichkeitsverteilung besser erkennen kann. Die Ordnung $r = 10$ teilt in diesem Fall q nicht, weshalb wir auch mit geringer Wahrscheinlichkeit Werte messen, die etwas weiter von $\frac{q}{r}$ entfernt sein können. Die Markierungen an der x-Achse zeigen die ganzzahligen Vielfachen von $\frac{q}{r}$ in diesem Bereich.

Diagramm bei dem wieder nur eine geringere Anzahl an Quantenbits gewählt wurde, um die Feinstruktur der Verteilung besser sichtbar zu machen (Abbildung 4.4), und ein Diagramm, welches die Verteilung angibt, wenn man q wie gefordert wählt (Abbildung 4.5).

Zum Abschluss fassen wir kurz die Resulate unserer Analyse zusammen:

Satz 4.3.15. *Die Wahrscheinlichkeit einen Wert $y \in \mathbb{Z}$ zu messen, der nahe eines ganzzahligen Vielfachen von $\frac{q}{r}$ liegt ist*

$$P(y) \geq \frac{1}{5r^2}.$$

Beweis. Wir haben bereits gesehen, dass wir einige Sonderfälle seperat behandeln können und die Wahrscheinlichkeit $P(y) \approx \frac{1}{r} > \frac{1}{5r^2}$ erhalten, falls

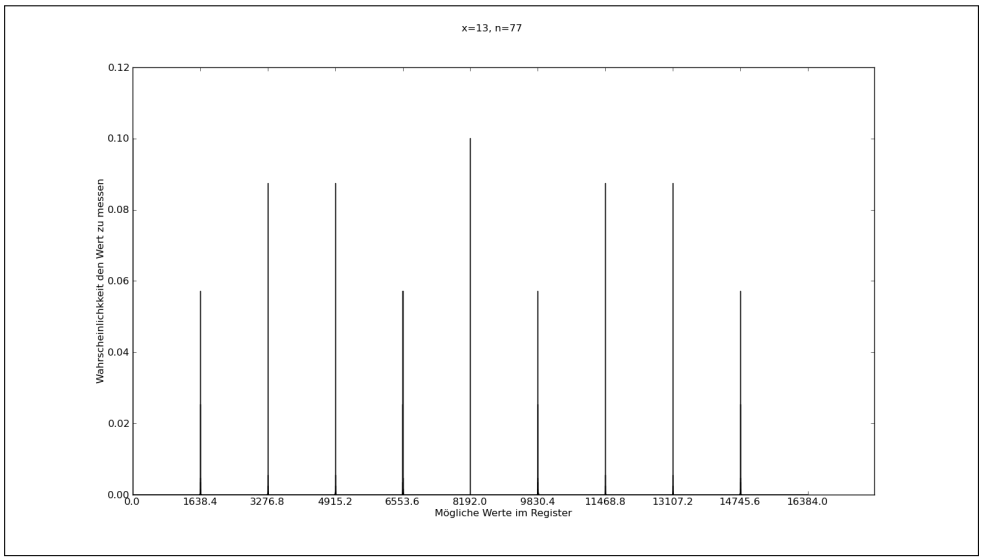

Abbildung 4.5: Wahrscheinlichkeitsverteilung nach Anwenden der Fouriertransformation, wenn wir q so wählen, wie vom Algorithmus vorgeschrieben. Man kann hier gut erkennen, dass die einzelnen Maxima der Verteilung in scharfen Peaks um die ganzzahligen Vielfachen von $\frac{q}{r}$ verteilt liegen.

n ausreichend groß gewählt wurde und somit $\mathcal{O}\left(\frac{1}{q}\right)$ vernachlässigt werden kann.
In diesem Fall konnten wir aber auch außerhalb der Sonderfälle $P(y)$ nach unten mit $\frac{1}{5r^2}$ abschätzen. □

4.3.3 Klassische Nachbereitung

Wie wir bereits im vorherigen Schritt gesehen haben, erhalten wir bei der Messung mit hoher Wahrscheinlichkeit einen Wert, der in der Nähe eines ganzzahligen Vielfachen von $\frac{q}{r}$ liegt.
Nehmen wir nun an

$$-\frac{r}{2} \leq ry \pmod{q} \leq \frac{r}{2},$$

dann exisitiert ein $d \in \mathbb{Z}$ mit

$$-\frac{r}{2} \leq ry - dq \leq \frac{r}{2}.$$

Dies können wir umformen zu

$$-\frac{1}{2q} \leq \frac{y}{q} - \frac{d}{r} \leq \frac{1}{2q}.$$

Sei nun $\frac{\tilde{d}}{\tilde{r}} \neq \frac{d}{r}$ so gewählt, dass

$$-\frac{1}{2q} \leq \frac{y}{q} - \frac{\tilde{d}}{\tilde{r}} \leq \frac{1}{2q},$$

dann ist

$$\left|\frac{\tilde{d}}{\tilde{r}} - \frac{d}{r}\right| \leq \frac{1}{q}.$$

Andererseits gilt

$$\left|\frac{\tilde{d}}{\tilde{r}} - \frac{d}{r}\right| = \left|\frac{\tilde{d}r - \tilde{r}d}{r\tilde{r}}\right| \geq \left|\frac{1}{r\tilde{r}}\right| > \frac{1}{n^2} \geq \frac{1}{q},$$

unsere Annahme, es könnte noch ein weiterer Bruch innerhalb dieses Intervalls liegen ist somit falsch. Wir können daher durch Berechnung des Näherungsbruchs aus $\frac{y}{q}$ die Ordnung von x abschätzen, da wir sichergestellt haben, dass nur ein einziger komplett gekürzter Bruch innerhalb dieses Intervalls liegt.

Bemerkung 4.3.16. *In der Simulation zeigt sich, dass es bei kleinen Eingaben reicht die Anzahl der Bits so zu wählen, dass $n \leq 2^L < 2 \cdot n$.*

Mit Hilfe der Kettenbruchapproximation berechnen wir einen Näherungsbruch zu $\frac{d}{r}$ aus unserem Messergebnis und erhalten aus dem Nenner dieses Bruchs die Ordnung r, falls d und r teilerfremd waren. Es gibt genau $\varphi(r)$ teilerfremde Zahlen $1 \leq d < r$. Jeder Bruch $\frac{d}{r}$ ist nahe bei einem Bruch $\frac{y}{q}$ mit oben angegebener Eigenschaft. Obendrein gibt es r verschiedene Werte für $x^k \pmod n$, da r die Ordnung von x ist. Insgesamt gibt es also $r \cdot \varphi(r)$ „gute" Werte und die Wahrscheinlichkeit einen solchen zu messen ist jeweils $\geq \frac{1}{5r^2}$ (siehe Satz 4.3.15).
Die Wahrscheinlichkeit r mit Hilfe dieses Algorithmus bestimmen zu können ist daher mindestens $\frac{\varphi(r)}{5r} \geq \frac{1}{5 \log r}$ (siehe hierzu [9] Anhang A.3). Wiederholt man den Quantenalgorithmus also $\mathcal{O}(\log r)$-mal, findet man die Ordnung mit Wahrscheinlichkeit $\mathcal{O}(1)$.
Tritt der Fall ein, dass d und r nicht teilerfremd sind, so scheitert unser Algorithmus. Man kann dieses Problem jedoch umgehen, indem man nicht nur überprüft, ob r die Ordnung von x ist, sondern auch $2 \cdot r, 3 \cdot r, \ldots$ testet. Es lässt sich zeigen, dass man mit diesem Schritt die Erfolgsquote eines einzigen Durchlaufs des Quantenalgorithmus auf $\mathcal{O}(1)$ bringen kann, wenn man die ersten $\mathcal{O}((\log n)^{1+\varepsilon})$ Vielfachen von r testet (siehe [23] S. 19).

Beispiel 4.3.17. *(Fortführung von Beispiel 4.3.11.)*
Messen wir nach der Fouriertransformation $y = 8$, erhalten wir

$$\frac{y}{q} = \frac{8}{16} = \frac{1}{2}.$$

Somit schätzen wir als Ordnung $\tilde{r} = 2$ und unser Algorithmus würde fehlschlagen. Testen wir zusätzlich noch $2 \cdot \tilde{r} = 4$ haben wir auch in diesem Fall die Ordnung ermitteln können.

Eine weitere Möglichkeit die Trefferquote des Algorithmus zu verbessern besteht darin nicht nur den im Quantenalgorithmus gemessenen Wert y zu testen, sondern auch $y \pm 1, y \pm 2, \ldots$, da auch diese Werte noch eine gewisse Wahrscheinlichkeit besitzen in der Nähe von $\frac{q}{r}$ zu liegen (vergleiche z.B. Abbildung 4.4).

4.4 Laufzeitanalyse

Unter gewissen Bedingungen können wir mit hoher Wahrscheinlichkeit die Ordnung eines Elementes mit Hilfe eines Quantencomputers berechnen. Sei nun im Folgenden $K := \log_2 n$.
Wir haben bereits gesehen, dass wir zur Durchführung des Algorithmus insgesamt $3K + 2K + 2$ Quantenbits benötigen, der Platzbedarf ist somit $\mathcal{O}(\log n)$ und steigt daher nur polynomiell an.
Betrachten wir die einzelnen Schritte im Algorithmus:

- Hadamardtransformation des ersten Registers erfordert $2K$ Hadamardgatter.
- $x^a \pmod n$ ist durchführbar mit $\mathcal{O}\left((\log n)^3\right)$ Quantengattern (siehe Satz 3.3.3).
- Messung des zweiten Registers benötigt K-Messungen.
- Quantenfouriertransformation ist realisierbar mit $\mathcal{O}\left((\log n)^2\right)$ Quantengattern (siehe Satz 3.4.5).
- Messung des ersten Registers benötigt $2K$ Messungen.

Der gesamte Quantenalgorithmus ist daher mit $\mathcal{O}\left((\log n)^3\right)$ Schritten durchführbar. Da auch alle Schritte im klassichen Teil mit polynomieller Laufzeit realisierbar sind haben wir einen effizienten Faktorisierungsalgorithmus gefunden.

Kapitel 5

Emulation des Algorithmus von SHOR

Inhaltsangabe

In diesem Kapitel gehen wir kurz darauf ein, wie man den Algorithmus von Shor auf einem klassischen Rechner umsetzen kann, um überprüfen zu können, wie gut das Verfahren funktioniert.
Zuerst betrachten wir, wie man ein Quantenregister abspeichern kann damit wir alle notwendigen Operationen auf Quantenregistern durchführen können. Danach stellen wir einige klassische Algorithmen vor, die wir zur klassischen Vor- und Nachbereitung des Algorithmus von Shor benötigen. Am Ende des Kapitels findet sich eine Kurzbeschreibung des Begleitprogramms zur Arbeit (siehe hierzu auch die dem Programm beiliegende Dokumentation) und wir betrachten einige Testdurchläufe dieses Programms, um einen Eindruck zu gewinnen wie erfolgreich der Algorithmus ist.

5.1 Das Quantenregister

Wie wir bereits in Kapitel 2.2 gesehen haben können wir ein n-Bit Quantenregister als Hintereinanderschaltung von Quantenbits auffassen. Man könnte also versuchen das Quantenregister als Array von Quantenbits zu implementieren. Offensichtliche Vorteile dieser Version wären

- Einfaches Anwenden von Operationen auf einzelne Quantenbits (z.B. Hadamardtransformation).

- Einfaches Messen einzelner Quantenbits und anschließende Transformation auf zulässige Werte.

Jedoch stößt man mit diesem Ansatz relativ schnell auf Probleme. Quantenbits können verschränkt sein und ein Register mit verschränkten Quantenbits kann man auf diese Art nicht mehr darstellen. Das Anwenden kontrollierter Operationen wäre daher auch nicht mehr durchzuführen.
Eine bessere Art der Darstellung ist es, das Quantenregister als Vektor des $\mathbb{C}^{2^n}$ auffassen. Der Zustand des Quantenregisters ist eindeutig bestimmt durch

$$|R\rangle = \sum_{i=0}^{2^n-1} a_i|i\rangle \quad \text{mit } a_i \in \mathbb{C},$$

wobei wir, wie in Kapitel 2.2 angegeben, die $|i\rangle$ mit den Basisvektoren des $\mathbb{C}^{2^n}$ auffassen. Wollen wir also den Zustand des Registers angeben genügt es die Paare (a_i, i) zu speichern.
Vorteile:

- Ist $a_i = 0$ müssen wir das Paar (a_i, i) nicht mit abspeichern. Daher ist der Bedarf an Arbeitsspeicher bei *dünnbesetztem Register* nicht besonders hoch.

- Kontrollierte Operationen wirken meist nicht auf die Amplituden a_i sondern auf den Zustandsvektor $|i\rangle$. Getrenntes Abspeichern ermöglicht uns also nur den entsprechenden Teil zu bearbeiten.

Ein Nachteil dieser Darstellung ist, dass es nun aufwendiger wird Operationen auf einzelnen Quantenbits durchzuführen. Konzeptionelle Schwierigkeiten erhalten wir mit dieser Darstellung jedoch nicht mehr.

5.2 Klassischer Teil des Algorithmus

Wie bereits erwähnt besteht der Algorithmus von Shor aus einem Quantenteil zur Berechnung der Ordnung einer Zahl und einem klassischen Teil mit dessen Hilfe wir sicherstellen, dass der Algorithmus durchgeführt werden kann.
Wichtig ist hierbei, dass die eingesetzten Algorithmen *effizient durchführbar* sein müssen, damit die durch den Quantenalgorithmus gewonnene Laufzeit nicht durch einen exponentiellen klassischen Anteil zerstört wird.

5.2.1 Schnelles Potenzieren

Zum *schnellen Potenzieren* benutzen wir die gleiche Idee, die wir schon verwendet haben um Potenzen mit einem Quantencomputer zu berechnen. Wir wandeln den Algorithmus allerdings in eine für klassische Rechner günstigere Form ab (vgl. [6] S. 38/39).

Algorithmus 5.2.1. *(square-and-multiply)*
Eingabe: $x \in \mathbb{Z}, n \in \mathbb{N}, N \in \mathbb{N}$
Ausgabe: $x^n \pmod N$

1. *Setze* $e := 1$
2. *Solange* $n > 0$*:*
   - *Falls* n *ungerade: Setze* $e := e \cdot x \pmod N$
   - *Setze* $x := x^2 \pmod N$
   - *Setze* $n := \left\lfloor \frac{n}{2} \right\rfloor$
3. *Ausgabe:* e

Satz 5.2.2. *Algorithmus 5.2.1 ist in polynomieller Zeit durchführbar und berechnet das gewünschte Ergebnis.*

Beweis. Wir betrachten zunächst das Innere der Schleife. $n := \left\lfloor \frac{n}{2} \right\rfloor$ können wir mit einem Bitshift realisieren und diesen Schritt daher bei der Analyse vernachlässigen. Es bleiben zwei Multiplikationen übrig, die wir jeweils mit Aufwand $\mathcal{O}\left((\log N)^2\right)$ abschätzen können, da x und e stets kleiner als N sind (vgl. [2] Proposition 2.4.1).
Wir benötigen $\mathcal{O}(\log n)$ Schleifendurchläufe und da in unserem Fall auch n kleiner als N gewählt wurde, erhalten wir somit für den kompletten Algorithmus eine Laufzeit von $\mathcal{O}\left((\log N)^3\right)$. □

5.2.2 Primzahltest

Das nächste Problem, das zu lösen ist, ist die Prüfung der Eingabe auf Primalität, da ein Faktorisierungsalgorithmus in diesem Fall keinen Sinn macht und nur unnötige Rechenzeit vergeudet.
Es sind jedoch effiziente Algorithmen bekannt um zu testen, ob eine Zahl prim ist, weshalb wir uns diesen Schritt erlauben können.
Ein *deterministischer Primzahltest* mit polynomieller Laufzeit wäre der *AKS-Test* (benannt nach den Findern M. Agrawal, N. Kayal und N. Saxena). Der Algorithmus basiert auf folgender Idee:

Satz 5.2.3. *Sei* $a \in \mathbb{Z}$ *und* $n \in \mathbb{N}$ *mit* $n \geq 2$ *und* $ggT(a, n) = 1$*. Dann ist* n *genau dann prim, wenn*

$$(x + a)^n = x^n + a \pmod n.$$

Beweis. Siehe [2] Proposition 6.3.1. □

Diese Gleichung zu verifizieren wäre jedoch noch zu aufwendig, weshalb man einen Reduktionsschritt einbauen muss um den Algorithmus effizient durchführen zu können (siehe hierzu [2] Proposition 6.3.6 oder [1] Abschnitt 4.). Setzt man dies in einen Algorithmus um, sieht man, dass der Test zwar in polynomieller Zeit durchführbar ist ($\mathcal{O}\left((\log n)^{21/2}\right)$ vgl. [1] Theorem 5.1), der Grundaufwand des Algorithmus ist jedoch so hoch, dass sich der Einsatz in der Praxis nicht lohnt.
Wir werden nun einen weiteren Algorithmus ansprechen, mit dem wir feststellen können, ob eine Zahl zusammengesetzt ist (vgl. hierzu auch [6] Kapitel 8.4). Es handelt sich hier jedoch um einen probabilistischen Algorithmus, daher gilt:

- Gibt der Algorithmus *zusammengesetzt* aus, dann ist die Eingabe eine zusammengesetzte Zahl.
- Gibt der Algorithmus *prim* zurück, dann könnte die Eingabe prim sein, könnte aber auch zusammengesetzt sein.

Der Algorithmus bietet aber eine Möglichkeit die Wahrscheinlichkeit dafür abzuschätzen, dass eine zusammengesetzte Zahl als prim erkannt wird, weshalb man durch mehrere unabhängige Durchläufe die Falsch-Positiv Wahrscheinlichkeit sehr klein machen kann.

Lemma 5.2.4. *Sei $n \in \mathbb{N}$ eine Primzahl > 2. Weiter seien*

$$s := \max\{r \in \mathbb{N} : 2^r \text{ teilt } n-1\}$$

und

$$d := \frac{n-1}{2^s},$$

sowie $a \in \mathbb{N}$ mit

$$ggT(a,n) = 1.$$

Dann gilt entweder

$$a^d = 1 \pmod{n} \tag{5.1}$$

oder es gibt ein $r \in \{0,1,\ldots,s-1\}$ mit

$$a^{2^r d} = -1 \pmod{n} \tag{5.2}$$

Beweis. Siehe [6] S. 127, Theorem 8.4.1. □

Ist n eine Primzahl, dann muss entweder (5.1) oder (5.2) erfüllt sein. Finden wir ein a, bei dem dies nicht der Fall sein sollte, haben wir sichergestellt, dass n keine Primzahl sein kann. Wir nennen a in diesem Fall einen *Zeugen gegen die Primalität von n.*

Lemma 5.2.5. *Ist $n > 2$ eine ungerade, zusammengesetzte Zahl. Dann gibt es in der Menge $\{1, 2, \ldots, n-1\}$ höchstens $\frac{n-1}{4}$ Zahlen, die zu n teilerfremd und keine Zeugen gegen die Primalität von n sind.*

Beweis. Siehe [6] S. 127, Theorem 8.4.3. □

Algorithmus 5.2.6. *(Miller-Rabin-Test)*
Eingabe: *$n \in \mathbb{N}$ mit $n > 3$ ungerade und $t \in \mathbb{N}$ die Anzahl der Durchläufe.*
Ausgabe: *„zusammengesetzt" oder „mit Wahrscheinlichkeit $1 - \frac{1}{4^t}$ prim"*

1. *Setze Zählervariable $c := 0$*
2. *Ziehe eine zufällige Zahl $a \in \{1, \ldots, n-1\}$*
3. *Teste die in Lemma 5.2.4 angegebenen Bedingungen*
   - *Falls(5.1) und (5.2) nicht erfüllt sind: Ausgabe: „zusammengesetzt"*
   - *sonst setze $c := c + 1$.*
4. *Falls $c < t$: Gehe zu 2.*
5. *Ausgabe: „wahrscheinlich prim"*

Im Begleitprogramm zur Arbeit wurde der Miller-Rabin-Test mit $t = 25$ verwendet. Die Falsch-Positiv Wahrscheinlichkeit ist demnach kleiner als

$$\frac{1}{2^{50}} \approx 8.9 \cdot 10^{-16}.$$

5.2.3 Primzahlpotenzen

Als letztes müssen wir noch einen Algorithmus bereitstellen mit dessen Hilfe wir feststellen können, ob eine Zahl $n \in \mathbb{N}$ mit $n > 1$ eine *Primzahlpotenz* ist.
Wie wir sehen werden, müssen wir uns nicht darauf beschränken Primzahlpotenzen zu finden, sondern können einen effizienten Algorithmus zum Finden von allgemeinen Potenzen angeben (siehe auch [2] Proposition 6.3.7). Dass wir damit zwar einige Eingaben mehr herausfiltern als eigentlich gewollt macht in diesem Fall nichts aus, da wir aus der Darstellung als Potenz die Faktorisierung einer Zahl ablesen können und somit nicht das Ergebnis des restlichen probabilistischen Algorithmus abwarten müssen.
Nehmen wir an n wäre in der Form $n = m^k$. Dann gilt

$$k = \log_m n \leq \log_2 n.$$

Die Idee hinter unserem Algorithmus ist nun, dass wir suchen, ob die Funktion

$$f(x) := x^k - n$$

für ein $k \in \{2, \ldots, \log_2 n\}$ eine ganzzahlige Nullstelle in $(0, n)$ besitzt. Wir verwenden hierzu eine modifizierte Form des Bisektionsverfahrens (vgl. auch [22] Kap. 4.2.1 und [11] Kap. 5.7).
Da $k > 1$ und $n > 1$, gilt

$$f(0) = 0 - n = -n < 0$$

und

$$f(n) = n^k - n = n \cdot (n^{k-1} - 1) > 0.$$

Die Funktion f, aufgefasst als Abbildung von $\mathbb{R} \to \mathbb{R}$, ist stetig und streng monoton wachsend. Deshalb gibt es genau eine Nullstelle zwischen 0 und n. Wir berechnen als nächstes $\tilde{m} := f\left(\lfloor \frac{x}{2} \rfloor\right)$ und erhalten somit 3 mögliche Fälle:

- $\tilde{m} < 0$: Die Nullstelle liegt im Intervall $\left(\lfloor \frac{x}{2} \rfloor, n\right)$
- $\tilde{m} = 0$: $\lfloor \frac{x}{2} \rfloor$ ist die gesuchte Nullstelle.
- $\tilde{m} > 0$: Die Nullstelle liegt im Intervall $\left(0, \lfloor \frac{x}{2} \rfloor\right)$

Wenden wir diesen Test sukzessive auf das jeweils neue Intervall an werden wir entweder eine ganzzahlige Nullstelle finden oder wir finden heraus, dass die Nullstelle in einem Intervall $I_k = (k, k+1)$ zu finden ist. In diesem Fall haben wir sichergestellt, dass die Nullstelle der Funktion keine ganze Zahl sein kann und können den nächsten Exponenten testen oder den Algorithmus mit der Meldung $n \neq m^k$ abbrechen, falls bereits alle möglichen Exponenten getestet wurden.
In der Realisierung wurden die Randpunkte jeweils dem Intervall hinzugefügt, obwohl sie keine Lösungen sein können. Als Abbruchbedingung betrachten wir daher den Fall, dass $\#I \leq 3$.

Algorithmus 5.2.7. *(Modifiziertes Bisektionsverfahren)*
Eingabe: *$n \in \mathbb{N}$ mit $n > 1$*
Ausgabe: *m, k mit $n = m^k$ falls n eine ganzzahlige Potenz ist.*

1. *Für k von 2 bis $\lfloor \log_2 n \rfloor$:*
   - *Setze $a := 0$.*
   - *Setze $b := n$.*
   - *Setze $I := [a, b] \subset \mathbb{Z}$.*

- *Setze* $l := \#I$.
- *So lange* $l > 3$*:*
  (a) *Setze* $m := a + \lfloor \frac{l}{2} \rfloor$.
  (b) *Setze* $\tilde{m} := m^k - n$.
  (c) *Falls* $\tilde{m} = 0$*:*
  Ausgabe: $n = m^k$.
  (d) *Falls* $\tilde{m} < 0$*: Setze*

$$a := m,\ I := [a, b],\ l := \#I.$$

  (e) *Falls* $\tilde{m} > 0$*: Setze*

$$b := m,\ I := [a, b],\ l := \#I.$$

- *Falls* $l = 3$*:*
  (a) *Setze* $m := a + 1$.
  (b) *Falls* $m^k = n$*:*
  Ausgabe: $n = m^k$.

2. *Ausgabe:* $n \neq m^k$.

5.3 Struktur des Begleitprogramms zur Arbeit

Ein Teil der Arbeit war, den Quantenalgorithmus auf einem klassischen Rechner zu simulieren. Die hierzu nötigen Hilfsmittel sind:

- Ein Zufallsgenerator, da wir bei der Messung von Quantenbits ein Zufallsexperiment durchführen müssen.

  Der Zufallsgenerator wurde in eine Hilfsklasse `Random` gekapselt (zu finden im Modul `__random.py`), die in den anderen Klassen verwendet wird. Dies hat den Vorteil, dass der zugrundeliegende Zufallsgenerator ausgetauscht werden kann (z.B. kann auf das Statistikprogramm R[1] umgestellt werden), ohne dass man an anderen Stellen den Generator austauschen muss.

- Ein Primzahltest im Modul `millerRabin.py`.

- Das Modul `classicAlgorithms.py` enthält eine Sammlung benötigter klassischer Algorithmen (euklidischer Algorithmus, schnelles Potenzieren, Kettenbruchapproximation, etc.). Mehr hierzu findet man in Abschnitt 5.2.

[1] http://www.r-project.org/

Für Quantenalgorithmen steht die Klasse `Quantumregister` zur Verfügung, welche im Modul `quantum_register.py` zu finden ist. Diese Klasse verwaltet die Zustände im Register und realisiert Operationen mit konstantem Aufwand (z.B. Hadamard-Transformation auf einem Bit, CNOT- und Toffoligatter). Die Anzahl der Quantenbits wird hier nur durch den verfügbaren Arbeitsspeichers begrenzt und es wird nicht kontrolliert, ob man auf ein ,,gültiges Bit" zugreift.
Für den praktischen Einsatz gedacht ist die im Modul `quantum_computer.py` enthaltene Klasse `Quantumcomputer`. Diese Klasse erlaubt einerseits, dass man das Register in verschiedene Teilregister aufspaltet, welche dann auch übersichtlicher ausgegeben werden können, andererseits ist auch eine Speicherverwaltung realisiert worden. So wird z.B. darauf geachtet, dass nicht mehr temporäre Bits verwendet werden, als dem Algorithmus erlaubt wurden, dass Keines doppelt verwendet wird und dass sie wieder ordnungsgemäß freigegeben werden. Obendrein enthält diese Klasse auch aufwendigere Operationen wie die Quantenfouriertransformation und arithmetische Operationen. Strukturell entspricht dies somit dem Aufbau dieser Arbeit, da wir auch hier aufwendigere Quantenoperationen von einfachen Registeroperationen getrennt hatten.
Zusätzlich existieren noch einige kleinere Anwendungsprogramme, mit denen man die in der Arbeit angegebenen Grafiken erzeugen kann:

- `probabilities.py`: Dieses Programm führt den Quantenalgorithmus durch. Nach der Quantenfouriertransformation wird das Register jedoch nicht mehr gemessen sondern es werden die Wahrscheinlichkeiten der einzelnen Messwerte geplottet.

- `classicshor.py`: Dieses Programm ist eine rein klassische Version des Algorithmus von Shor. Die Ordnung wird hier nicht mit Hilfe des Quantenalgorithmus bestimmt, sondern mittels eines klassischen Verfahrens (exponentieller Aufwand). Das Programm dient dazu Referenzgrafiken zu erstellen, damit man die Erfolgsquote des Quantenalgorithmus einschätzen kann.

- `generate_plots.py`: Dieses Programm startet den Algorithmus von Shor (wahlweise die rein klassiche Version oder die Version mit Quantenalgorithmus) für alle Zahlen $n \in \{0, ..., 99\}$ und gibt am Schluss ein Diagramm aus, welches die Fehlerquoten visualisiert. Man kann sich mit Hilfe dieses Diagramms einen groben Überblick über die Erfolgsaussichten verschaffen. Zusätzlich diente dieses Programm auch dazu ,,gutartige" und ,,bösartige" Werte zu finden, welche dann als Beispielmessungen in dieser Arbeit verwendet wurden.

- `shor.py`: In diesem Programm ist der Algorithmus von Shor (inkl. simuliertem Quantenalgorithmus) realisiert. Die klassische Nachbearbei-

tung zur Ermittlung der Ordnung wurde hierbei in eine eigene Funktion ausgelagert, damit man leichter sehen kann ab welcher Stelle der klassische Rechner wieder die Arbeit übernehmen muss. In diesem Programm wurde eine Zeitmessung eingebaut, mit deren Hilfe die durchschnittliche Rechenzeit der einzelnen Quantenoperationen gemessen wird.

Man sollte beachten, dass der Quantenalgorithmus nur mit exponentiellem Aufwand auf einen klassischen Rechner umgesetzt werden konnte. Als grobe Richtlinie lässt sich sagen, dass bei jedem zusätzlich verwendetem Quantenbit der benötigte Arbeitsspeicher verdoppelt wird und daher auch moderne Rechner sehr schnell an ihre Grenzen stoßen. Auch die Rechenzeit wächst sehr schnell an, wobei man in der Simulation sieht, dass die arithmetischen Operationen mit Abstand die meiste Rechenzeit benötigen.

5.4 Testmessungen

Wir haben in Kapitel 4 bereits gesehen, dass der Algorithmus aus einem Quantenteil besteht, mit dessen Hilfe man die Ordnung eines Elements bestimmt und aus einem klassischen Teil, der möglicherweise aus dieser Ordnung einen Teiler ermitteln kann. Der Speicherbedarf verdoppelt sich jedoch mit jedem zusätzlichen Quantenbit in etwa und auch die Rechenzeit zur Durchführung des Algorithmus wird immer größer, daher können nur kleine Eingaben getestet werden. Da es sich obendrein um einen probabilistischen Algorithmus handelt, können wir nicht aus einer einzigen Messung auf den Gesamterfolg des Algorithmus schließen. Um die Erfolgswahrscheinlichkeiten abschätzen zu können muss man deshalb immer mehrere Messungen durchführen. Es hat sich gezeigt, dass sich die Erfolgsquote des Algorithmus bei 10 000 Durchläufen relativ gut stabilisiert, weshalb diese Grenze für die folgenden Messungen gewählt wurde.
Außerdem konnten wir feststellen, dass sowohl im klassischen Teil eine gewisse Unsicherheit besteht ein Ergebnis zu erhalten, z.B wenn die Ordnung des zufällig gezogenen Elements ungerade ist, als auch im Quantenteil.
Um die Erfolgsquote des Quantenteils zu schätzen, ersetzen wir diesen durch ein deterministisches Verfahren zur Bestimmung der Ordnung und vergleichen die Ergebnisse dieses Verfahrens mit denen unseres simulierten Quantenalgorithmus.
Abbildungen 5.1, 5.3, 5.5 und 5.7 sind die Diagramme zu unserer deterministischen Referenzmessung, während in den Abbildungen 5.2, 5.4, 5.6 und 5.8 Messungen dargestellt sind, die mit Hilfe des simulierten Quantenalgorithmus durchgeführt wurden. Die beiden Diagramme 5.9 und 5.10 zeigen die Fehlerhäufigkeiten des Algorithmus für alle Zahlen kleiner als 100 auf, wobei man einerseits sehr gut sehen kann, dass der Quantenalgorithmus eine sehr hohe Trefferquote hat, da kein Unterschied zu den Ergebnissen aus der

Referenzmessung erkennbar ist und andererseits auch insgesamt die Erfolgswahrscheinlichkeit einen Faktor zu finden wesentlich besser ist, als die in der Abschätzung in Satz 4.3.5 angegebenen 50%.

Abbildung 5.1: Referenzmessung zur Zerlegung von $n = 15$.

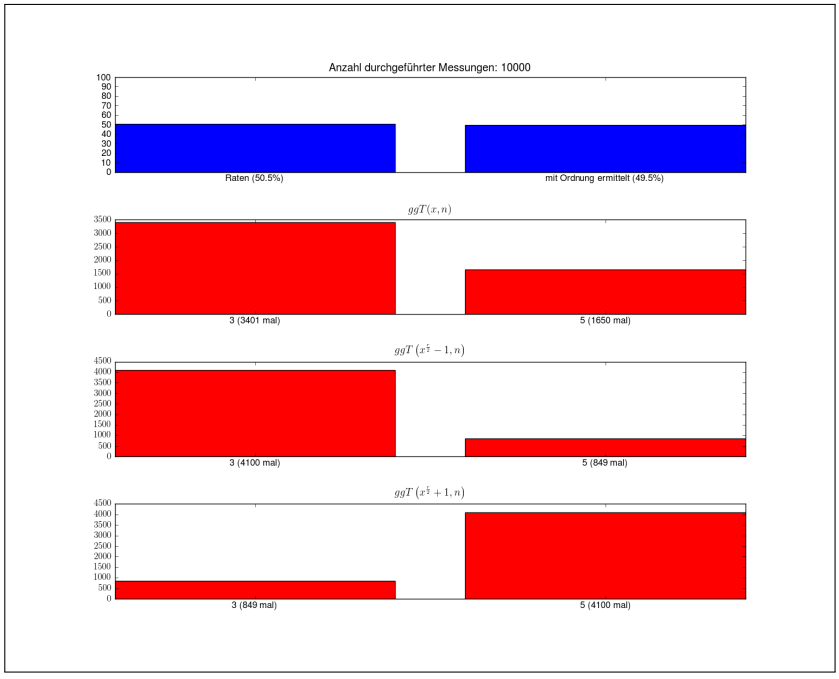

Abbildung 5.2: Messung zur Zerlegung von $n = 15$.

Abbildung 5.3: Referenzmessung zur Zerlegung von $n = 77$.

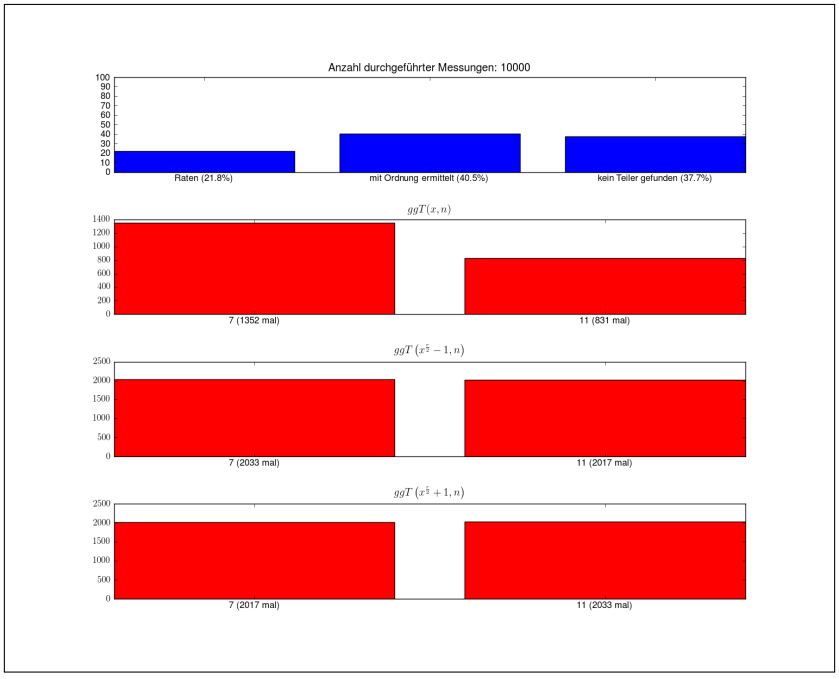

Abbildung 5.4: Messung zur Zerlegung von $n = 77$.

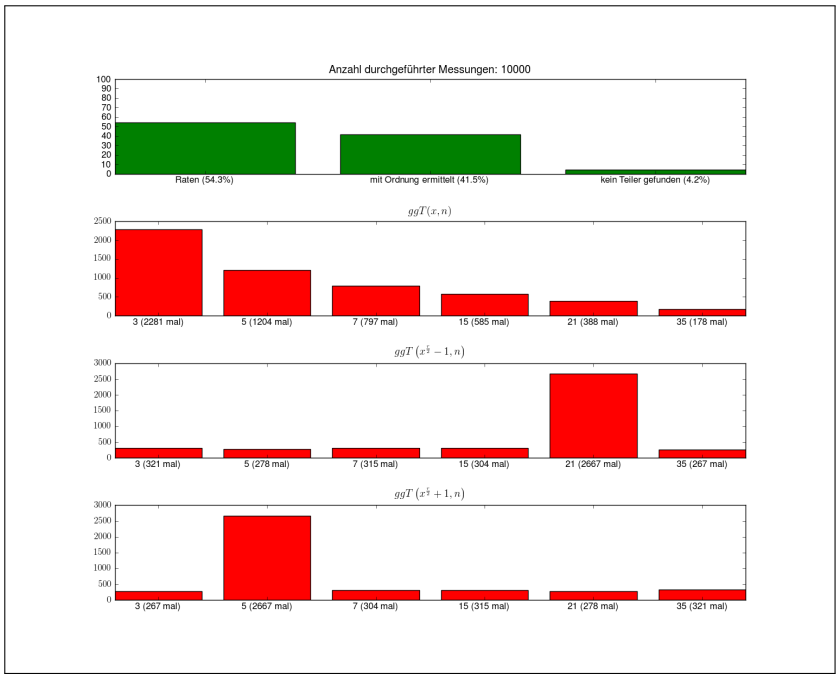

Abbildung 5.5: Referenzmessung zur Zerlegung von $n = 105$.

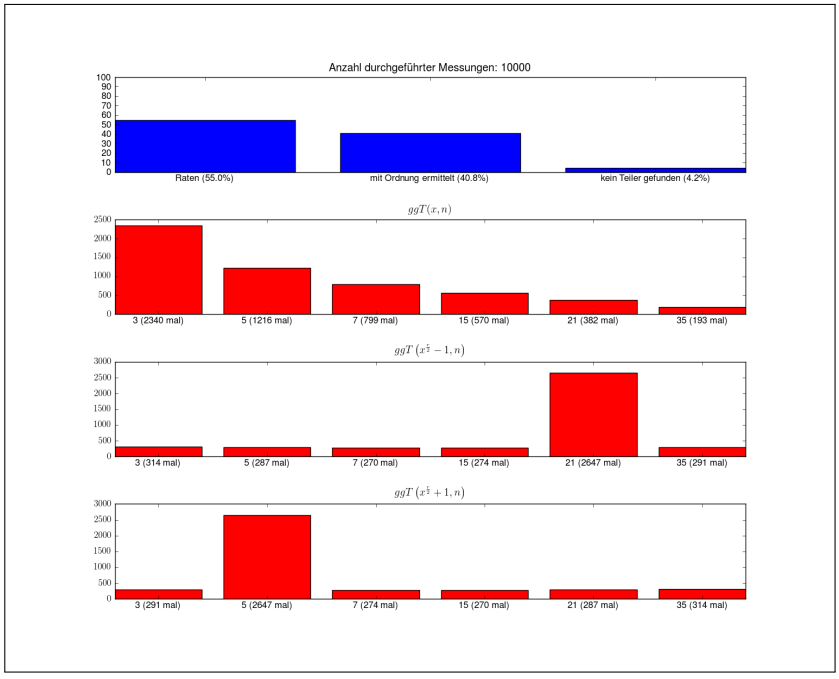

Abbildung 5.6: Messung zur Zerlegung von $n = 105$.

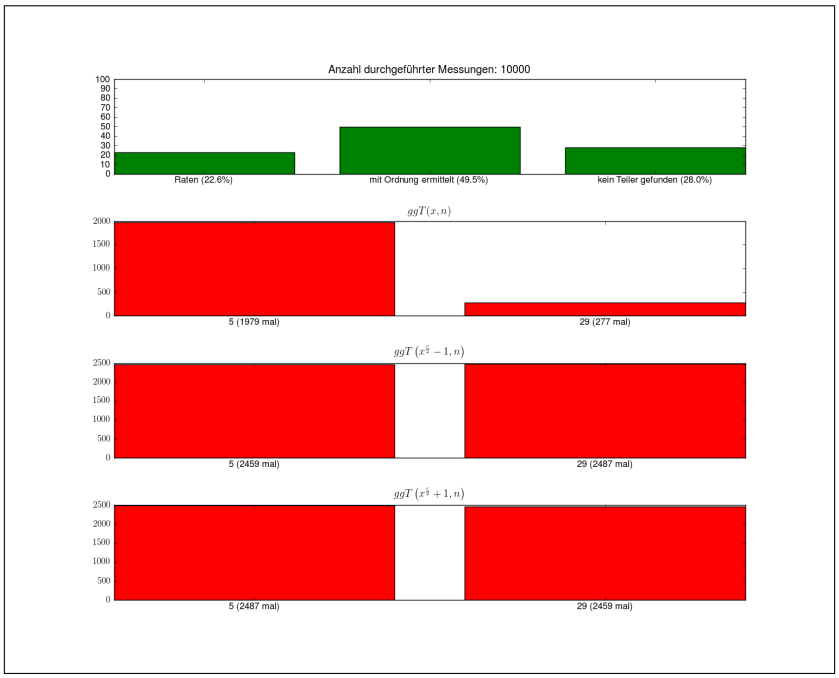

Abbildung 5.7: Referenzmessung zur Zerlegung von $n = 145$.

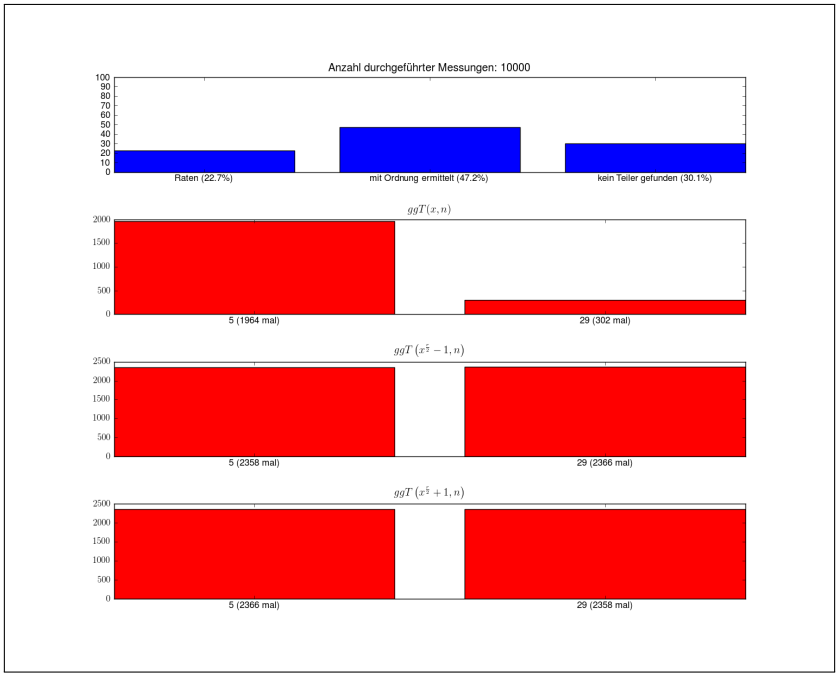

Abbildung 5.8: Messung zur Zerlegung von $n = 145$.

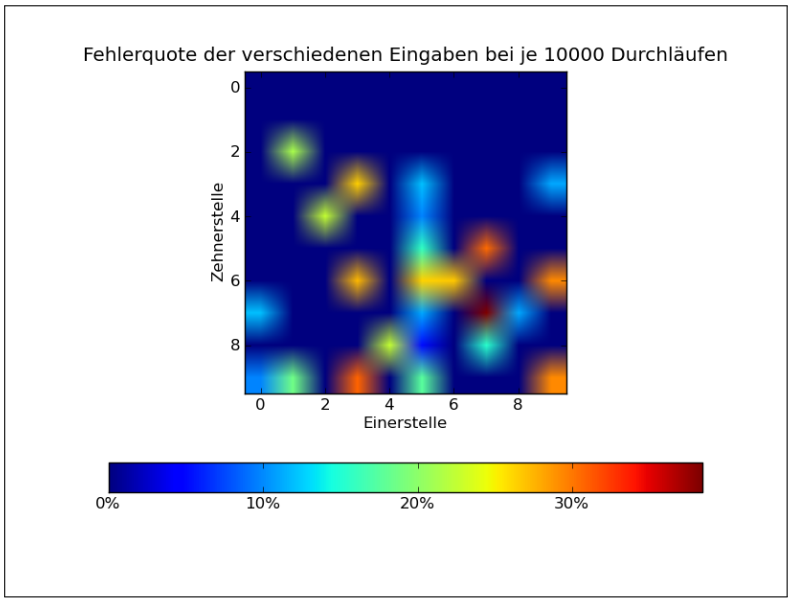

Abbildung 5.9: Visualisierung der Fehlerhäufigkeit des Algorithmus bei der Zerlegung aller Zahlen zwischen 0 und 99. Verwendet wurde hier der Algorithmus mit deterministischer Berechnung der Ordnung.

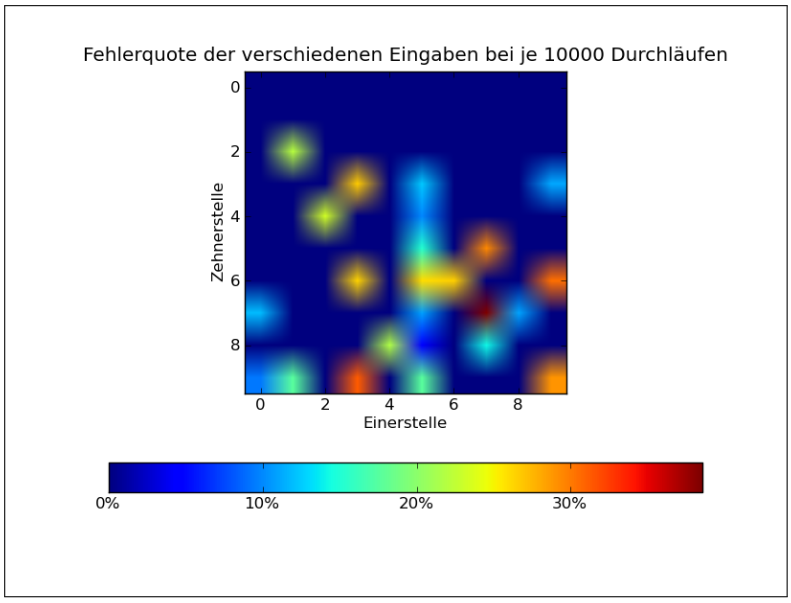

Abbildung 5.10: Visualisierung der Fehlerhäufigkeit des Algorithmus bei der Zerlegung aller Zahlen zwischen 0 und 99. Verwendet wurde hier der Quantenalgorithmus inklusive klassischer Nachbearbeitung zur Verbesserung der Trefferwahrscheinlichkeit.

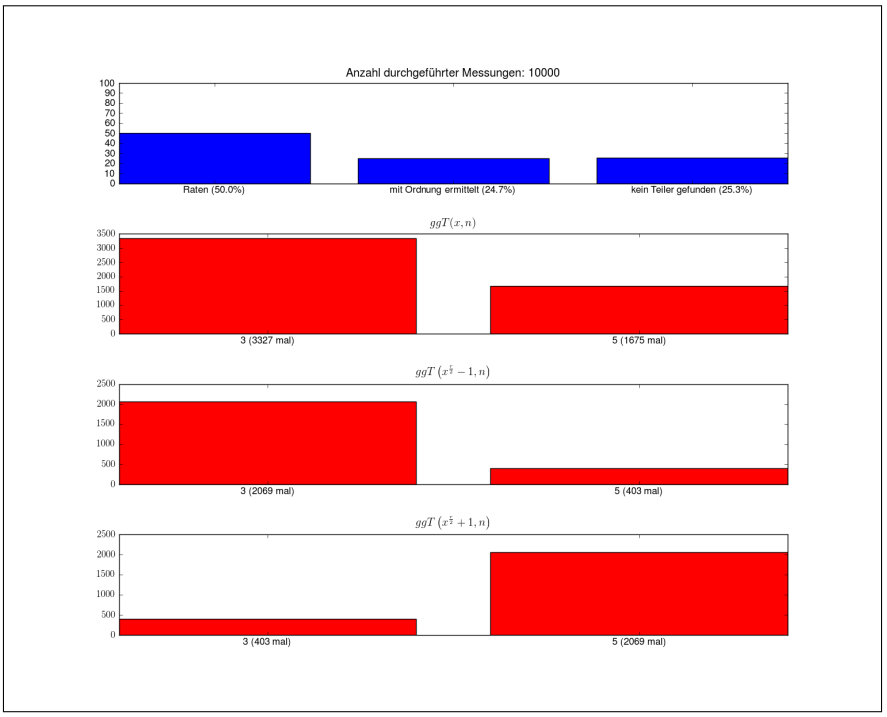

Abbildung 5.11: Messung zur Zerlegung von $n = 15$ ohne Nachbearbeitung des Ergebnisses.

Wir haben obendrein noch festgestellt, dass eine gewisse klassische Nachbearbeitung für unseren Quantenalgorithmus nötig ist, damit wir mit hoher Wahrscheinlichkeit die Ordnung bestimmen zu können. Mittels unserer Simulation können wir illustrieren, dass dieser Nachbearbeitungsschritt eine sinnvolle Investition darstellt, obwohl noch einige Rechenzeit beansprucht wird.
Führen wir eine Messung ohne die klassische Nachbearbeitung durch erhalten wir beispielsweise das in Abbildung 5.11 gezeigte Resultat. Verglichen mit dem Durchlauf in Abbildung 5.2, welcher die Nachbearbeitung enthält, sieht man, dass die Erfolgsquote des Quantenalgorithmus hier sehr viel schlechter ausfällt. Dies liegt aber auch daran, dass der Quantenalgorithmus nur einmal durchlaufen wird und nicht $\mathcal{O}(\log r)$-mal, was eigentlich nötig wäre um eine hohe Erfolgswahrscheinlichkeit garantieren zu können.
Zuletzt wollen wir noch kurz die in Satz 4.3.9 angegebene Verbesserung des klassischen Teils betrachten.
Wir haben bereits gesehen, dass unser Algorithmus bei $n = 77$ eine sehr hohe Fehlerquote aufweist (ca. 38%, siehe auch Abbildung 5.4). Bauen wir jedoch noch diesen zusätzlichen Test mit ein erhalten wir das in Abbildung 5.12 angegebene Bild und sehen, dass sich der Algorithmus durch diesen Trick deutlich verbessert hat.

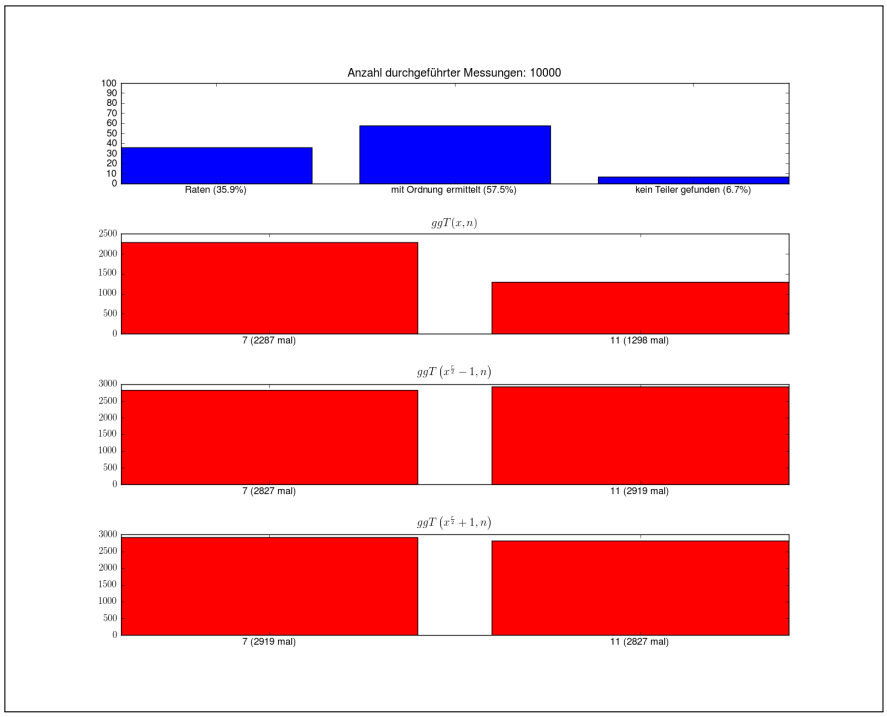

Abbildung 5.12: Messung zur Zerlegung von $n = 77$ mit Auswahlregel nach Satz 4.3.9.

Anhang A

Python als Programmiersprache

Zur Simulation des Quantencomputers wurde Python als Programmiersprache gewählt, da eine große Anzahl nützlicher Bibliotheken zur Umsetzung des Problems existiert.
Wichtig waren hierbei einerseits die Numerikbibliothek Numpy[1], die einige wichtige mathematische Funktionen und den Zufallsgenerator bereitstellt, andererseits die Plotbibliothek Matplotlib[2], mit deren Hilfe die Diagramme in den Anwendungsprogrammen erzeugt werden.
Da Python eine interpretierte Sprache ist, ist das Programm zwar nicht sonderlich schnell, Python bietet jedoch die Möglichkeit Teile des Codes in „schnellere" Sprachen (z.B. C/C++ oder Fortran) auszulagern. Um plattformunabhängig zu bleiben wurde jedoch hierauf verzichtet.

[1] http://numpy.scipy.org/
[2] http://matplotlib.sourceforge.net/

Anhang B

libquantum

Statt eine Eigenentwicklung zur Simulation des Quantenalgorithmus zu nutzen hätten wir auch auf die von Björn Butscher und Hendrik Weimer entwickelte C-Bibliothek libquantum[1] zurückgreifen können, bei der unter Anderem bereits alle nötigen Quantengatter und auch der Algorithmus von Shor bereits implementiert sind.
Es handelt sich hier jedoch um eine sehr ausführliche Bibliothek, die einiges mehr behandelt, als eigentlich zur Durchführung des Algorithmus von Shor notwendig ist. Beispielsweise ignoriere ich in meiner Simulation Dekohärenzeffekte und nehme an, dass Quantenbits nicht ohne Weiteres den Zustand ändern, wohingegen dies in libquantum berücksichtig werden kann. Will man den Algorithmus von Shor auch mit größeren Eingaben testen wäre es ratsam die Testprogramme auf diese Bibliothek zu portieren oder zumindest Teile des Programmes nach C, C++ oder Fortran auslagern.

[1] http://www.libquantum.de/

Anhang C

Technische Umsetzung

Bisher haben wir immer nur angegeben, dass die Schritte im Quantencomputer alle durchführbar sind, haben aber noch nicht klar gestellt, ob es sich hierbei womöglich nur um ein theoretisches Konstrukt handelt. Der folgenden Abschnitt soll nun einen kleinen Einblick in die technische Realisierung von Quantencomputern und speziell in die Umsetzung von Quantenbits geben.

C.1 Photonen als Quantenbit

In diesem Abschnitt betrachten wir, wie wir mit Hilfe von Photonen einen kleinen Quantencomputer bauen und den in Algorithmus 3.1.1 angegebenen Zufallsgenerator realisieren können.

Die hier zugrundeliegende Idee ist, dass man Photonen, die mit Hilfe einer *Einzelphotonenquelle* erzeugt worden sind, auf einen *Strahlteiler* schickt. *Detektoren* messen, ob das Photon am Strahlteiler *transmittiert* (interpretiert als $|0\rangle$) oder *reflektiert* (interpretiert als $|1\rangle$) wurde.

Der Strahlteiler entspricht in diesem Versuch unserem Hadamardgatter.

Dieser Versuchsaufbau kann auch als Schülerexperiment[1] realisiert werden, wie der Lehrstuhl ,,Didaktik der Physik" an der ,,Friedrich-Alexander-Universität Erlangen-Nürnberg" demonstriert. Die Abbildung C.1, C.2 und C.3 stammen aus der Versuchsanleitung dieses Schülerversuchs

[1] siehe: http://www.didaktik.physik.uni-erlangen.de/quantumlab/index.html

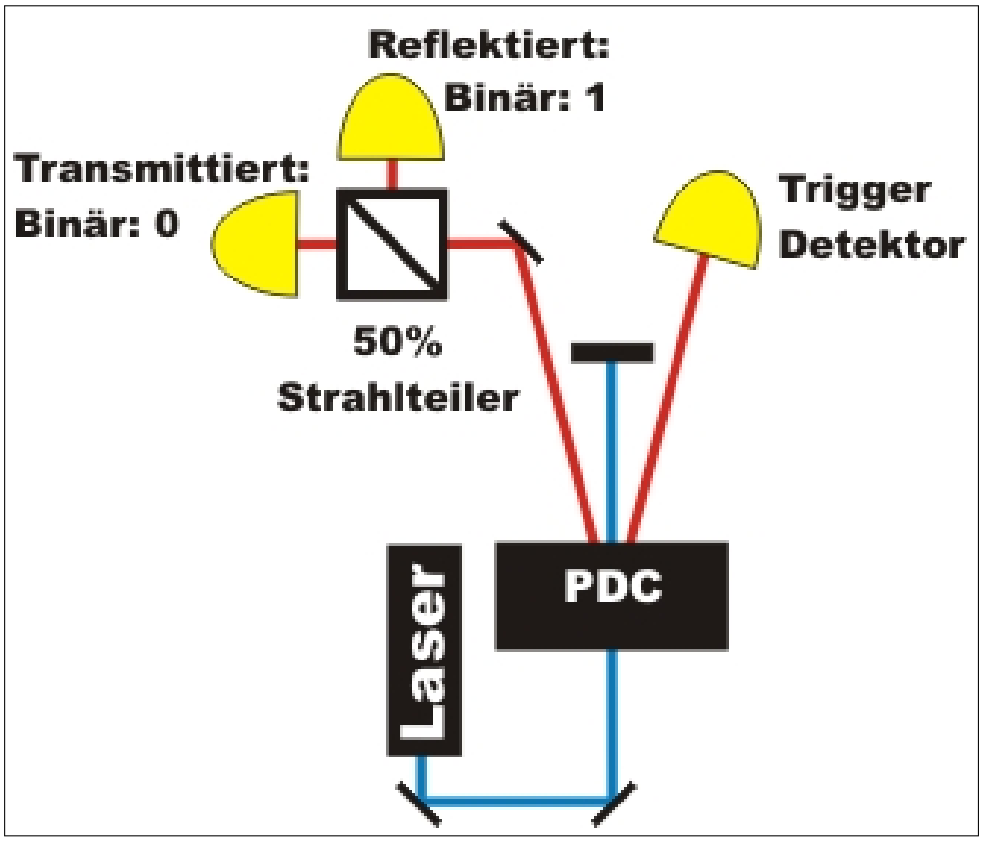

Abbildung C.1: Versuchsaufbau zur Realisierung eines Quantenzufallsgenerators.

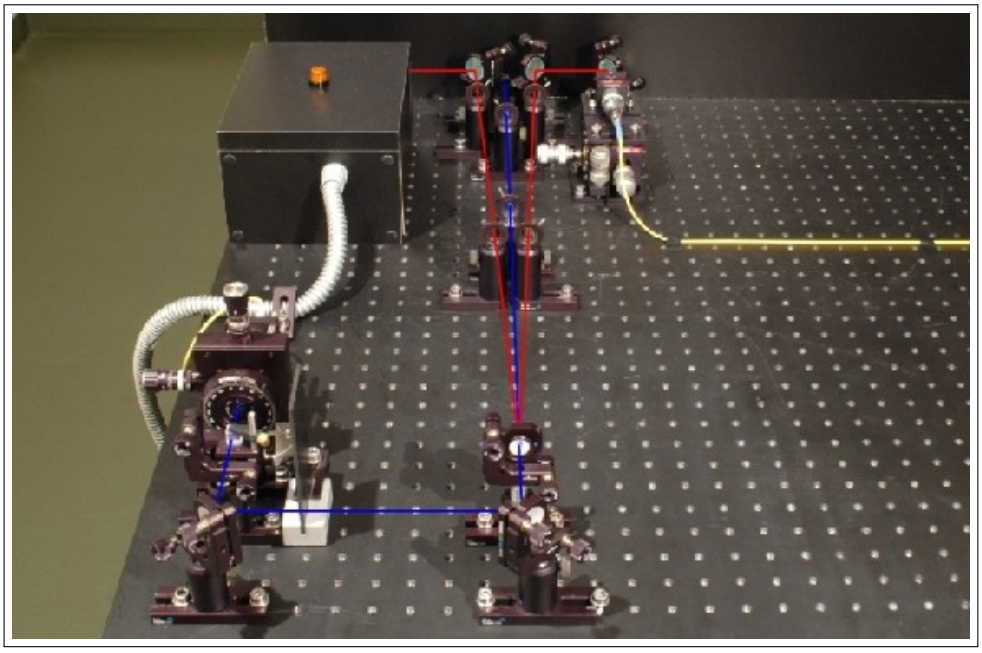

Abbildung C.2: Versuchsaufbau zur Realisierung einer Einzelphotonenquelle. Der Detektor links dient zur Triggerung der beiden Detektoren im Quantenzufallsgenerator (siehe Abbildung C.3).

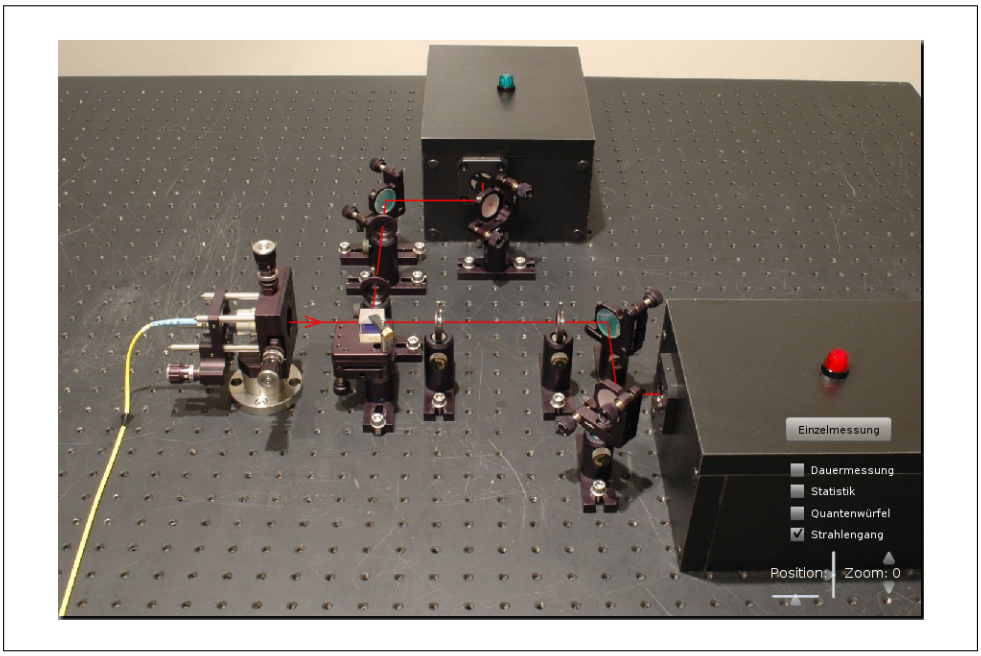

Abbildung C.3: Versuchsaufbau zur Realisierung des Quantenzufallsgenerators mit eingezeichnetem Strahlengang. Auf der Homepage zu diesem Versuch kann der Ablauf des Zufallsgenerators mittels einer Flashanimation betrachtet werden.

C.2 Ionenfallen

Eine weitere Möglichkeit Quantenbits zu realisieren ist, dass man *Ionen* in sogenannten *Ionenfallen* einsperrt. Die Ionen können in diesem Fall gewisse Energieniveaus einnehmen. Der Grundzustand entspricht dann z.B. $|0\rangle$ und ein angeregter Zustand entspricht $|1\rangle$.
Nachteile dieses Ansatzes sind, dass die Ionen sehr stark gekühlt werden müssen, damit keine Dekohärenz auftritt. Operationen auf Quantenbits können jedoch mit Hilfe von Laserpulsen ausgeführt werden, weshalb dieser Ansatz ein möglicher Kandidat für die Realisierung größerer Quantenrechner sein könnte. In [4] Kapitel VI. wird die Anzahl der nötigen Laserpulse zur Durchführung arithmetischer Operationen analysiert und zum Abschluss noch gezeigt, dass der Quantenalgorithmus für $n = 15$ mit 38 Laserpulsen durchführbar ist.

C.3 Kernspinresonanz

Als letztes werden wir noch *Kernspinresonanz* (engl.: *nuclear magnetic resonance*) zur Realisierung von Quantencomputer betrachten, da dieses Verfahren eingesetzt wurde, um den ersten offiziell bestätigten Quantencomputer

zu bauen, der den Algorithmus von Shor erfolgreich ausführen konnte[2]. Hierbei wurde mit Hilfe von 7 Quantenbits die Zahl 15 zerlegt, indem eine leicht modifizierte Variante des Algorithmus von Shor verwendet wurde (siehe hierzu [24]).

[2]IBM Research Division (2001, December 20). IBM's Test-Tube Quantum Computer Makes History; First Demonstration Of Shor's Historic Factoring Algorithm. ScienceDaily. Retrieved March 23, 2011, from http://www.sciencedaily.com/releases/2001/12/011220081620.htm

Literaturverzeichnis

[1] Agrawal, Manindra, Kayal, Neeraj und Saxena, Nitin. "PRIMES is in P". URL `http://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf`.

[2] Baldoni, Maria Welleda, Ciliberto, Ciro und Piacentini Cattaneo, Giulio Maria. *Elementary Number Theory, Cryptography and Codes*. Springer-Verlag Berlin Heidelberg, 1. Auflage 2009.

[3] Barenco, Adriano, Bennett, Charles H., Cleve, Richard, DiVincenzo, David P., Margolus, Norman, Shor, Peter, Sleator, Tycho, Smolin, John A. und Weinfurter, Harald. "Elementary gates for quantum computation". *Phys. Rev. A*, 52, 5:3457–3467 1995. doi:10.1103/PhysRevA.52.3457. URL `http://arxiv.org/abs/quant-ph/9503016`.

[4] Beckman, David, Chari, Amalavoyal N., Devabhaktuni, Srikrishna und Preskill, John. "Efficient networks for quantum factoring". *Phys. Rev. A*, 54, 2:1034–1063 1996. doi:10.1103/PhysRevA.54.1034. URL `http://arxiv.org/abs/quant-ph/9602016v1`.

[5] Bronstein, I. N., Semendjajew, K. A., Musiol, G. und Mühlig, H. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 5. Auflage 2001.

[6] Buchmann, Johannes. *Einführung in die Kryptographie*. Springer-Verlag Berlin Heidelberg, 4. Auflage 2008.

[7] Deuflhard, Peter und Hohmann, Andreas. *Numerische Mathematik 1*. Walter de Gruyter GmbH & Co. KG, 4. Auflage 2008.

[8] Deutsch, D., Ekert, A. und Lupacchini, R. "Machines, Logic and Quantum Physics". *ArXiv Mathematics e-prints* 1999. URL `http://arxiv.org/abs/math/9911150`.

[9] Ekert, Artur und Jozsa, Richard. "Quantum computation and Shor's factoring algorithm". *Rev. Mod. Phys.*, 68, 3:733–753 1996. doi:10.1103/RevModPhys.68.733.

[10] Fließbach, Torsten. *Quantenmechanik*. Spektrum Akademischer Verlag, 4. Auflage 2005.

[11] Freund, Roland W. und Hoppe, Ronald H. W. *Stoer/Bulirsch: Numerische Mathematik 1*. Springer-Verlag Berlin Heidelberg, 10. Auflage 2007.

[12] Grover, L. K. "A fast quantum mechanical algorithm for database search". *ArXiv Quantum Physics e-prints* 1996. URL `http://arxiv.org/abs/quant-ph/9605043`.

[13] Homeister, Matthias. *Quantum Computing verstehen*. Friedr. Vieweg & Sohn Verlag, 2. Auflage 2008.

[14] Horowitz, Paul und Hill, Winfield. *The Art of Electronics*. Cambridge University Press, 2. Auflage 1989.

[15] Huber, M. und Plesch, M. "Purification of genuine multipartite entanglement". *ArXiv e-prints* 2011.

[16] Lanzagorta, Marco und Uhlmann, Jeffrey. "Quantum Computer Science". *Synthesis Lectures on Quantum Computing*, 1, 1:1–124 2008. doi:10.2200/S00159ED1V01Y200810QMC002. URL `http://www.morganclaypool.com/doi/abs/10.2200/S00159ED1V01Y200810QMC002`.

[17] Leander, Gregor. "Improving the Success Probability for Shor's Factoring Algorithm". *ArXiv Quantum Physics e-prints* 2008. URL `http://arxiv.org/abs/quant-ph/0208183v1`.

[18] Müller-Stach, Stefan und Piontkowski, Jens. *Elementare und algebraische Zahlentheorie*. Friedr. Vieweg & Sohn Verlag, 1. Auflage 2006.

[19] Noh, C., Chia, A., Nha, H., Collett, M. J. und Carmichael, H. J. "Quantum Teleportation of the Temporal Fluctuations of Light". *Physical Review Letters*, 102, 23:230.501–+ 2009. doi:10.1103/PhysRevLett.102.230501.

[20] Preskill, John. "Lecture Notes for Physics 229: Quantum Information and Computation" 1998. URL `http://www.theory.caltech.edu/people/preskill/ph229/`.

[21] Prof. Dr. Frank Lempio. "Vom Abakus zum Quantencomputer". Vorlesungsskript Sommersemester 2009.

[22] Schwarz, Hans Rudolf und Köckler, Norbert. *Numerische Mathematik*. B.G. Teubner Verlag / GWV Fachverlage GmbH, 6. Auflage 2006.

[23] Shor, Peter W. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". *SIAM J.SCI.STATIST.COMPUT.*, 26:1484 1997. URL `http://arxiv.org/abs/quant-ph/9508027v2`.

[24] Vandersypen, L. M. K., Steffen, M., Breyta, G., Yannoni, C. S., Cleve, R. und Chuang, I. L. "Experimental Realization of an Order-Finding Algorithm with an NMR Quantum Computer". *Physical Review Letters*, 85:5452–5455 2000. doi:10.1103/PhysRevLett.85.5452. URL `http://arxiv.org/abs/quant-ph/0007017`.

[25] Vedral, Vlatko, Barenco, Adriano und Ekert, Artur. "Quantum networks for elementary arithmetic operations". *Phys. Rev. A*, 54, 1:147–153 1996. doi:10.1103/PhysRevA.54.147. URL `http://arxiv.org/abs/quant-ph/9511018v1`.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Neusorg, den 29. März 2011

Wolfgang Riedl