Realisierung arithmetischer Schaltkreise auf dem Einweg-Quantencomputer

Markus Bleicher

12. Mai 2003

vergeben von Prof. Dr. Martin Wirsing
betreut von Dr. Hans J. Briegel

Cluster states sind die Grundlage einer möglichen Implementierung von Quantencomputern. Dabei handelt es sich um Zustände aus vielen Teilchen, die einen hohen Verschränkungsgrad aufweisen. Sie können z.B. mit in optischen Gittern gefangenen ultrakalten Atomen herstellt werden, die man durch einen interferometrischen Prozess in einen verschränkten Zustand versetzt.

Auf solchen Zuständen lässt sich durch Messung jedes Quantengatternetzwerk über dem universellen Satz aus dem CNOT-Gatter und beliebigen Rotationen auf einem Qubit darstellen. Das Messen der Cluster ist also ein universales Berechnungsmodell.

Die Zeitkomplexität für dieses "Rechnen durch Messen" ist in praktisch allen Fällen niedriger als die der Netzwerke, da man Messungen auch für hintereinander geschaltete Gatter parallel ausführen kann. Beispielsweise braucht das Standard-Quantenfouriertransformationsnetzwerk für eine n Qubit Eingabe 2n-1 Schritte, während das entsprechende Messprogramm für einen Cluster nur n Messzeitpunkte benötigt. Netzwerke, die nur aus CNOT-, Hadamard- und $\pi/2$ -Gattern bestehen brauchen sogar nur einen Messschritt, d.h. alle Messungen zur Realisierung eines solchen können gleichzeitig ausgeführt werden.

Die kritische Ressource des Einwegquantencomputers ist der Platz, da normalerweise das ganze Netzwerk auf einen Clusterzustand passen muss und man auch nicht eine Reduzierung der Platzanforderung durch eine Verschlechterung der Zeitkomplexität erreichen kann.

Der Algorithmus von Peter Shor zur Faktorisierung von natürlichen Zahlen ist im Moment die bedeutendste Entdeckung auf dem Gebiet der Quanteninformatik. Seine Implementierung auf einem Quantencomputer zerfällt im Wesentlichen in zwei Teile: Die Quantenfouriertransformation und ein arithmetisches Netzwerk zur Berechnung einer modularen Exponentiation.

Inhaltsverzeichnis

1 Einleitung

Bei einem Quantencomputer werden die Berechnungszustände nicht auf ein klassisches deterministisches physikalisches System abgebildet, sondern auf ein Quantensystem(, das normalerweise durch einen Hilbertraum beschrieben wird). Der wesentliche Unterschied ist, dass n 2-Zustandsteilchen (bits) im klassischen Fall einen Raum (über dem Körper {0,1}) ihrer möglichen Konfigurationen von der Größe n aufspannen, während das Quantensystem mit n 2-Zustandsteilchen (qubits) sich in einem Hilbertraum der Dimension 2^n bewegt.

Ein Beispiel: Ein 3-Bit-Register hat die 2^3 verschiedenen Zustände 000, 001, 010, 011, ..., 111, während ein 3-qubit-Register sich in den Zuständen $\alpha_0 |000\rangle + \alpha_1 |001\rangle + \alpha_2 |010\rangle + \alpha_3 |011\rangle + ... + \alpha_7 |111\rangle$ befinden kann, also einer "Überlagerung" der "reinen" Registerzustände. Dieses Phänomen der "Quantenparallelität" versucht man auszunutzen, wobei das Problem darin besteht, dass sich gewünschte Ergebnisse nicht einfach auslesen lassen, sondern bei einer Messung die $|\alpha_i|^2$ die Wahrscheinlichkeit angeben, das entsprechende Resultat auch zu messen. Bei naiver Vorgehensweise kann man also exponentiell viele Berechnungen gleichzeitig durchzuführen, die möglicherweise einzige richtige Antwort erhält man aber auch nur mit exponentiell geringer Wahrscheinlichkeit.

Das ist eine bisher nur an sehr vereinzelten Beispielen gelöstes Frage. Zu nennen sind neben Problemen von theoretischem Interesse wie Deutsch, Deutsch-Jozsa, Simon der Suchalgorithmus von Grover und vor allem die Faktorisierungs- und Logarithmierungsalgorithmen von Shor.

Neben diesen theoretisch-algorithmischen, sind auch die praktisch-experimentellen Schwierigkeiten gewaltig: Die Dekohärenz - das Phänomen, dass die für einen Quantencomputer benötigte unitäre Entwicklung nur in völliger Abgeschlossenheit von den Umwelt stattfindet und diese Abgeschlossenheit quasi natürlich durch Umwelteinflüsse wie Rauschen aufgehoben wird - verhindert gegenwärtig, dass man einen Quantencomputer über größere Zeiträume laufen lassen kann. Die Problematik des Rauschens gibt es auch in klassischen Systemen, nur ist sie da viel weniger grundsätzlich: Zum einen kollabiert normalerweise nicht das gesamte System, wenn es an einer Stelle gestört wird (eine Eigenschaft, die gleichzeitig die Stärke und die Schwäche des Quantensystems ausmacht: positiv lässt sich durch Manipulation einer Stelle das ganze System transformieren, was Deutsch "Quantenparallelismus" getauft hat, ein Begriff, den ich für die technischen Ausprägungen der Systeme übernehme, negativ zerstören schon geringste ungewollte Ankopplungen an die Umgebung das ganze Rechenschema meist vollständig), zum anderen ist es die Idee der Digitalisierung, die Fehler unterhalb einer gewissen Schranke einfach unterdrückt. Etwas zumindest vom Ergebnis her ähnliches wird mit dem Einsatz von Quantenfehlerkorrektur versucht: Man erklärt bestimmte Systemzustände von vornherein als fehlerhaft und projiziert das Gesamtsystem immer wieder in den nächstliegenden

"richtigen" Zustand. Beim Digitalschaltkreis macht das jedes Gatter, das seinen Ausgang auf volle Spannung schaltet, wenn der Eingang eine gewisse Grenze über- (oder unter- je nach gewünschter Schaltung) schreitet. Beim Quantensystem löst man das z.B. so, dass man die eigentlichen Berechnungszustände mit fehlerkorrigierenden Codes auf die realen abbildet und mit diesen rechnet und an adäquaten Stellen auf den hoffentlich richtigen Code eines Berechnungszustandes projiziert.

Das Verhältnis zwischen der Zeit- und Speicherplatzkomplexität klassischer Computer und Quantencomputer ist für interessante Probleme - also solche, die ausserhalb von P liegen - ungeklärt. Bewiesen ist eine bessere Zeitkomplexität meines Wissens nur für ein paar künstliche Probleme sowie für die Quantenfouriertransformation und für den Suchalgorithmus von Grover, die alle in $O(n \log n)$ liegen. Die Aufklärung in dieser Frage ist natürlich erheblich dadurch erschwert, dass die Ergebnisse der klassischen Komplexitätstheorie nach wie vor nur einzelne Zusammenhänge zeigen und es keine Systematik gibt, sowie Fragen wie P=NP ungeklärt sind. Auch für die, wie der Shor-Algorithmus gezeigt hat, in BQP (bounded error quantum polynomial: also der Klasse von Problemen, die auf einem Quantenrechner in polynomieller Zeit mit einer bestimmten, festen Wahrscheinlichkeit gelöst werden können) liegenden Probleme der Faktorisierung, des diskreten Logarithmus oder allgemeiner der Suche nach versteckten Gruppen ist unbekannt, ob sie nicht auch in (BP)P liegen. Da sie sowohl in NP als auch in co-NP liegen, gehören sie nicht zu den schwierigsten Problemen dieser Klassen.

Der einzige bekannte Äquivalenz zwischen einer Komplexitätsklasse für Quantencomputer und einer, die nicht auf Quanten bezug nimmt, ist $\mathrm{NQP} = \mathrm{co} - \mathrm{C_{=}P}$, wobei NQP die Klasse der auf einem nichtdeterministischen Quantencomputer in polynomieller Zeit lösbaren Probleme ist, ein Analogon zur Klasse NP für klassische Computer ist und $\mathrm{C_{=}P}$ die Klasse der Probleme ist, für die eine nichtdeterministische Turingmaschine die gleiche Anzahl von akzeptierenden und ablehnenden Pfaden produziert.

Dass die bewiesenen Komplexitätsunterschiede zwischen klassischen und Quantencomputern gegenwärtig noch spärlich sind, ist jedoch kein Argument für die Einschätzung, dass sie vielleicht als physikalisches Anschauungsmaterial für das Verschränkungsphänomen dienen können, aber aus informationstechnischer, besonders komplexitätstheoretischer Sicht, keine Relevanz hätten. Denn ein Berechnungsverfahren ist - auch wenn seine Überlegenheit nicht bewiesen ist - natürlich ein Fortschritt, wenn es gegenüber den bekannten Alternativen schneller ist.

Eine deutlichere Schwierigkeit als die eventuell nicht vorhandene größere Rechenkraft sind die Probleme beim Bau eines Quantenrechners. Es sind einige Durchbrüche nötig, um von den auch theoretisch schlecht skalierenden Systemen, die man bisher entworfen und teilweise gebaut hat, zu ernstzunehemend großen Computern zu kommen. Es ist gut möglich, dass unüberwindliche Schwierigkeiten auftreten und das Modell des Quantencomputers ein Modell bleibt.

Ein ungewöhnlicher Ansatz in diesem Zusammenhang ist der Einweg-Quantencomputer von Robert Raussendorf und Hans J. Briegel [11], der zumindest das theoretische Potenzial hat, als relevant großes System implementiert zu werden, da er auf zwar schwierigen, aber gut erforschten und experimentell beherrschten Grundlagen aufbaut, und eine mögliche Schwachstelle, die Implementierung von unitären Transfor-

mationen und das Erzeugen von Verschränkungen wärend des Berechnungsprozesses, inhärent ausschliesst.

Bei der Komplexitätsbetrachtung dieses Rechners zeigt sich, dass sowohl die zeitliche Anordnung, also das Hintereinanderschalten, als auch die räumliche Ausdehung, also die Zahl der Registerqubits, bei der Abbildung von Quantengatternetzwerken auf Einweg-Quantencomputer ihre Entsprechung in der räumlichen Struktur der Messungen hat, die Raumkomplexität auf dem Einweg-Quantencomputer also ein Produkt der Raum- und Zeitkomplexität des Quantenschaltkreises ist, während seine Zeitkomplexität normalerweise der Tiefe des Quantenschaltkreises entspricht.

In dieser Arbeit werden Schaltkreise auf dem Einweg-Quantencomputer implementiert, die zusammen eine Ausführung des Shor-Algorthmus zur Faktorisierung ermöglichen sollen. Dazu benötigt man die gut untersuchte Quantenfouriertransformation und die weniger gut untersuchten arithmetischen Schaltkreise. Das Ziel ist eine Vertiefung des Verständnisses vor allem der "Einweg"-Implementierung des Paradigmas "Rechnen mit Quanten", weniger praktische Fragen, die mir beim gegenwärtigen Stand der Forschung auch irrelevant erscheinen.

Dabei werden die bisher für den Einweg-Quantencomputer entworfenen Gatter erfasst und größere Strukturen aus ihnen assembliert, sowie ihre Raumkomplexität - das vermutlich für einen Einweg-Quantencomputer kritische Maß - abgeschätzt.

2 Was sind Quantencomputer?

Ein Computer ist eine Maschine, die Rechenoperationen nach einem Programm ausführt, d.h. der Zustand des Computers ändert sich in Abhängigkeit vom Programm, den eingegebenen Daten und seinem bisherigen Zustand.

Bei einem digitalen Computer besteht die Menge der möglichen Zustände - der Konfigurationsraum K - aus Tupeln über einem endlichen Alphabet Σ. Bei einem binären Computer ist $\Sigma = \{0, 1\}$. Bei einem n bit Computer ist $|K| = 2^n$.

Quantencomputer haben demgegenüber einen exponentiell größeren Konfigurationsraum, also bei n qubits (und k zugelassen Amplituden) $|K| = k^{(2^n)}$. Auf diesen Zustände lassen sich auch noch "klassische" Operationen (genauer: invertierbare Funktionen $\{0,1\}^n -> \{0,1\}^n$) gleichzeitig ausführen, ein Phänomen, dass man "Quantenparallelismus" nennt. Dem stehen jedoch zwei Nachteile gegenüber: Die (unproblematische) Notwendigkeit, dass alle Übergänge des Quantensystems unitär, d.h. aus programmtechnischer Sicht invertierbar, sein müssen. Und die Schwierigkeit, dass man die Zustände aus K nicht betrachten, nicht messen kann. Sie stellen immer nur Wahrscheinlichkeitsverteilungen dar. Das erfordert ganz andere Programmkonzepte.

2.0.1 Beispiel

Ein 3-Bit-Register hat die Zustände 000, 001, 010, 011, ..., 111, während ein 3-qubit-Register sich in den Zuständen $\alpha_0 |000\rangle + \alpha_1 |001\rangle + \alpha_2 |010\rangle + \alpha_3 |011\rangle + ... + \alpha_7 |111\rangle$ befinden kann, also einer "Überlagerung" der Registerzustände (die gleich den Basiszuständen der Rechenbasis des quantenmechanischen Systems sind).

Hat man nun eine unitäre Transformation - also ein Programm - mit der Eigenschaft $|b_1 b_2 b_3\rangle \rightarrow |b_1 b_2 (b_1 \oplus b_2) \oplus b_3\rangle$(lineare Abbildungen sind durch die Funktionswerte der Basiszustände eindeutig bestimmt) mit $\oplus$ = XOR und wendet sie auf den Basiszustand $\sum_{b_1,b_2 \in \{0,1\}} 1/\sqrt{4} |b_1 b_2 0\rangle$an, werden "quantenparallel" alle 4 2-bit-Additionen (modulo 2) ausgeführt. Die Ergebnisse lassen sich nun nicht einfach beliebig auslesen, sondern bei einer Messung ist $|\alpha_i|^2$ die Wahrscheinlichkeit, das entsprechende Resultat auch zu messen, hier misst man also die vier Zustände $|b_1 b_2 (b_1 \oplus b_2)\rangle$mit Wahrscheinlichkeit je 1/4.

Bei naiver Vorgehensweise kann man also exponentiell viele Berechnungen gleichzeitig durchzuführen, die möglicherweise einzige richtige Antwort erhält man aber auch nur mit exponentiell geringer Wahrscheinlichkeit.

2.0.2 Deutschs XOR Problem

Ein Algorithmus, der ein Problem schneller löst als auf einem klassischen Rechner möglich, ist der von Deutsch 1985 vorgestellte und von Cleve, Ekert, Macchiavello und Mosca 1998 verbesserte Algorithmus zu folgendem einfachen Problem:

Gegeben ein Orakel $f : \{0,1\} \to \{0,1\}$ gilt es zu entscheiden, ob es ausgeglichen ($f(0) \oplus f(1) = 1$) oder konstant ($f(0) \oplus f(1) = 0$) ist.

Dazu benötigt ein klassischer Algorithmus zwei Aufrufe von f.

Der Quantenalgorithmus, dem f als unitäre Transformation U_f mit $U_f |b_1 b_2\rangle = |b_1(f(b_1) \oplus b_2)\rangle$, also $U_f |00\rangle = |0f(0)\rangle$, $U_f |10\rangle = |1f(1)\rangle$, $U_f |01\rangle = |0(f(0) \oplus 1)\rangle$, $U_f |11\rangle = |1(f(1) \oplus 1)\rangle$, gegeben ist, wertet U_f auf einer Überlagerung von $|0\rangle$und $|1\rangle$aus.

$$
\begin{array}{ll}
\to^{H_2} & \\
= & 1/4(|00\rangle - |01\rangle + |10\rangle - |11\rangle) \\
\to^{U_f} & 1/4(|0f(0)\rangle - |0(f(0) \oplus 1)\rangle + |1f(1)\rangle - |1(f(1) \oplus 1)\rangle) \\
= & 1/4((-1)^{f(0)}(|00\rangle - |01\rangle) + (-1)^{f(1)}(|10\rangle - |11\rangle)) \\
= & 1/4((-1)^{f(0)} |0\rangle \otimes (|0\rangle - |1\rangle) + (-1)^{f(1)} |1\rangle \otimes (|0\rangle - |1\rangle)) \\
= & 1/4((-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle) \otimes (|0\rangle - |1\rangle)) \\
= & (-1)^{f(0)}/4(|0\rangle + (-1)^{f(1)-f(0)} |1\rangle) \otimes (|0\rangle - |1\rangle)) \\
= & (-1)^{f(0)}/4(|f(1) \oplus f(0)\rangle_x \otimes |1\rangle_x)
\end{array}
$$

Messen von qubit 1 in der x-(dualen)Basis liefert also die Antwort (was äquivalent ist zu einer Hadamard-Transformation mit anschliesenden Messen in der Rechen(z-)basis).

Frage: Ist U_f stärker als f, da ja 4 Funktionswerte bestimmt wurden (auch die, wo das Ergebnisqubit nicht leer war, und die eigentlich beliebig sein müssten) und nicht nur 2? U_f ist die einzig mögliche unitäre Vervollständigung. Ermöglicht die Umkehrung von Funktionen "for free" einfachere Berechnungen? Nein, klassisch lässt sich das Problem mit U_f auch nur mit zwei Funktionsaufrufen lösen wie mit f.

2.1 Die Beschreibung quantenmechanischer Systeme

Notation: $\overline{a}$ ist die konjugiert komplexe Zahl zu a.

Die **Zustandsräume** quantenmechanischer Systeme sind Teilmengen von Hilberträumen über $\mathbb{C}$ mit der **Norm 1**.

Ein **Hilbertraum** H ist ein **Vektorraum** mit einem **Skalarprodukt** $\langle .|. \rangle$ (mit den Eigenschaften $\langle \phi|\psi\rangle = \overline{\langle \phi|\psi\rangle}$, $\langle \phi|\phi\rangle \geq 0$ und $\langle c_1\phi_1 + c_2\phi_2|\psi\rangle = c_1 \langle \phi_1|\psi\rangle + c_2 \langle \phi_2|\psi\rangle$) und der Norm $\|\phi\| := \sqrt{\langle \phi|\phi\rangle}$, der **vollständig** ist, d.h. jede Cauchy-Folge $\{\phi\}_{i \in N} \subset H$ mit $\lim_{i,j\to\infty} \|\phi_i - \phi_j\| = 0$ konvergiert gegen ein ϕ mit $\lim_{i\to\infty} \|\phi_i - \phi\| = 0$.

Der Dualraum H^*des Hilbertraums H besteht aus den stetigen linearen Funktionalen $f : H \to \mathbb{C}$, wobei für jede Abbildung f ein eindeutiger Vektor $\phi_f = \sum_{b \in B} f(b)b$ existiert mit $f(\chi) = \langle \phi_f|\chi\rangle$ (lineare Funktionen stimmen überein gdw sie auf einer Basis übereinstimmen) für alle χ. Mit der Umkehrabbildung $\langle \phi| (\chi) = \langle \phi|\chi\rangle$ gilt $\forall\chi.\phi_{\langle\chi|} = 1_H$ und $\forall f. \langle \phi_f| = 1_{H^*}$ sowie $\phi_{af+bg} = a\phi_f + b\phi_g$ und $\langle a\phi + b\psi| = a \langle \phi| + b \langle \psi|$.

Die Elemente von H werden in der Diracschen Bra(c)Ket-Notation $|\phi\rangle$ geschrieben (Ket-Vektoren), in der von Neumann-Notation schlicht ϕ, die Objekte $|\phi\rangle$und ϕ sind also identisch. Die Elemente von H^* schreibt man $\langle \phi|$ (Bra-Vektoren).

Zwei physikalisch bedeutsame Beispiele für Hilberträume sind

$$l_2 = \{\{x\}_{i\in N} \subset \mathbb{C} | \sqrt{\sum_{i\in N} |x_i|^2} < \infty\}$$

mit dem Skalarprodukt $\langle x|y\rangle = \sum_{i\in N} x_i\overline{y_i}$, für die **Heisenbergsche Matrizenmechanik** und

$$L_2 = \{f : \mathbb{R} \to \mathbb{C} | \int |f(x)|^2\, dx\}$$

mit dem Skalarprodukt $\langle f|g\rangle = \int f(x)\overline{g(x)}dx$ mit dem die **Schrödingersche Wellenmechanik** arbeitet.

Das **Tensorprodukt** auf zwei Hilberträumen H_1 und H_2 ist ein Hilbertraum H und eine bilineare Abbildung $\otimes : H_1 \times H_2 \to H$ wird eine Realisierung des Tensorprodukts auf H_1 und H_2 genannt, falls das Skalarprodukt $\langle x \otimes y|x' \otimes y'\rangle = \langle x|x'\rangle_1 \langle y|y'\rangle_2$ für alle $x, y \in H_1$ und alle $x', y' \in H_2$ gilt und die Menge $\{x \otimes y|x \in H_1, y \in H_2\}$ dicht in H liegt. So eine Realisierung existiert immer und ist eindeutig bis auf Isomorphie[1]. Zusätzlich sollen die Tensorprodukte verträglich sein, d.h. $(A \otimes_{H_1} B)(a \otimes_{H_2} b) = (Aa) \otimes (Bb)$.

Im endlichen Fall auf zwei Matrizen $A^{l\times m} \otimes B^{n\times p}$ ist das Tensorprodukt eine Matrix $\mathbb{C}^{ln\times mp}$, wobei die $c_{i,j} = a_{\lceil i/n\rceil,\lceil j/p\rceil} b_{(i-1) \bmod l+1,(j-1) \bmod m+1}$, also mit

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{l1} & \cdots & a_{lm} \end{pmatrix} \text{ und } B = \begin{pmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{l1} & \cdots & b_{lm} \end{pmatrix} \text{ ist } C = \begin{pmatrix} a_{11}B & \cdots & a_{1m}B \\ \vdots & \ddots & \vdots \\ a_{l1}B & \cdots & a_{lm}B \end{pmatrix}$$

Die Abbildung ist, wie man nachrechnen kann, bilinear. Das Bild liegt (da $H = H_A \otimes H_B$ endlich) dicht in H.

Das Skalarprodukt wird wie üblich als $\langle (a_{hk})|\, (b_{xy})\rangle = \sum_{(i,j) \text{ mit } i<ln, j<mp} \overline{a_{ij}} b_{ij}$ definiert. Es ist eine hermitesche Form und erfüllt $\langle x \otimes y|x' \otimes y'\rangle = \langle x|x'\rangle_1 \langle y|y'\rangle_2$ für alle $x, y \in H_1$ und alle $x', y' \in H_2$ mit den üblichen Salarprodukten für H_1 und H_2.

Genaugenommen definiert diese Abbildung nicht ein Tensorprodukt, sondern für jedes Paar von Hilberträumen $A^{l\times m}$ und $B^{n\times p}$ je eines. Diese sind verträglich, und damit ein ordentlich definiertes Tensorprodukt.

Lineare Operatoren auf dem Hilbertraum H sind die linearen Funktionen $H \to H$. Jeder Operator (auf einem abzählbaren Hilbertraum) kann als eine Matrix dargestellt werden: Ist $B = \{|\gamma_i\rangle\, |i \in I\}$ eine Basis von H, dann ist $\langle\gamma_i|\, A\, |\gamma_j\rangle$ das i, j-te Element der Matrixdarstellung von A.

Für den **adjungierten** Operator A^*von A gilt $\langle A^*\phi|\varphi\rangle = \langle\phi|A\varphi\rangle$ für alle ϕ, φ. Ein Operator A ist **selbstadjungiert**, falls $A^* = A$. Die Matrixdarstellung von selbstadjungierten Operatoren ist hermitisch, $A = A^\dagger$, d.h. $a_{i,j} = \overline{a_{j,i}}$.

Unitäre Transformationen werden durch Matrizen mit der Eigenschaft $MM^\dagger = M^\dagger M = I$ dargestellt. Sie sind also invertierbar und ihre Inverse ist wieder unitär.

[1] nach [3], S.55

Projektoren P_{H_u} auf einen Unterraum H_u von H bilden den Vektor auf das eindeutige "Teilstück" ab, das in H_u liegt. Es gilt: $\phi = P_{H_u}\phi + P_{H_u^\perp}\phi$.

Die λ und $|\phi\rangle$ die die **Eigenwertgleichung** $A|\phi\rangle = \lambda|\phi\rangle$ des Operators A erfüllen, heissen seine Eigenwerte und Eigenvektoren.

Selbstadjungierte Operatoren besitzen eine so genannte **Spektraldarstellung**. Sind $\lambda_1, \ldots, \lambda_k$ die Eigenwerte des Operators A und P_i Projektoren auf die von den zugehörigen Eigenvektoren aufgespannten Unterräumen, so ist $A = \sum_{i=1}^{k} \lambda_i P_i$.

Algebraische Umformungen mit den obigen Elementen

$|a\phi + b\varphi\rangle = a|\phi\rangle + b|\varphi\rangle$,

$|a\phi + b\varphi\rangle = a\phi + b\varphi$,

$A|a\phi + b\varphi\rangle = a|A\phi\rangle + b|A\varphi\rangle$, ket-Vektoren sind nur Schnörkel in der Notation, man kann sie einfach weglassen, wenn klar ist, dass ein Hilbertraumvektor gemeint ist.

$\langle a\phi + b\psi| = \overline{a}\langle\phi| + \overline{b}\langle\psi|$, nach Definition des bra-Operators.

$A(|\varphi\rangle\langle\psi|) = (A|\varphi\rangle)\langle\psi|$, da $A(|\varphi\rangle\langle\psi|)\chi = A(|\varphi\rangle(\langle\psi|\chi)) = A((\langle\psi|\chi)|\varphi\rangle) = (\langle\psi|\chi)A|\varphi\rangle = (A|\varphi\rangle)(\langle\psi|\chi) = ((A|\varphi\rangle)\langle\psi|)\chi$.

$\phi \otimes (a\varphi + b\chi) = a(\phi\otimes\varphi) + b(\phi\otimes\chi)$,

$(a\varphi + b\chi)\otimes\phi = a(\varphi\otimes\phi) + b(\chi\otimes\phi)$, da $\otimes$bilinear ist.

$\langle\varphi\otimes\chi|\psi\otimes\phi\rangle = \langle\varphi|\psi\rangle\langle\chi|\phi\rangle$, nach Definition von $\otimes$.

$(A\otimes B)(\varphi\otimes\chi) = A\varphi\otimes B\chi$, nach Definition von $\otimes$.

$\langle\phi\otimes\varphi| = \langle\phi|\otimes\langle\varphi|$, da $\langle\phi\otimes\varphi|(\chi\otimes\omega) = \langle\phi\otimes\varphi|\,\chi\otimes\omega\rangle = \langle\phi|\,\chi\rangle\langle\varphi|\,\omega\rangle$ und $(\langle\phi|\otimes\langle\varphi|)(\chi\otimes\omega) = \langle\phi|\,\chi\otimes\langle\phi|\,\omega = \langle\phi|\,\chi\rangle\otimes\langle\varphi|\,\omega\rangle = \langle\phi|\,\chi\rangle\langle\varphi|\,\omega\rangle$.

In dieser Arbeit werden - wie bei der Beschreibung von Quantencomputern meist - nur abstrakte Hilberträume endlicher Dimension verwendet, auf die ich mich im folgenden beschränke.

Notation: $A^{(i)}$ meint die Anwendung des Operators A auf den durch i repräsentierten Unterraum des Hilbertraumes H, z.B. meint $\sigma_x^{(5)}$ eine σ_x-Drehung auf dem 5.Qubit. Exakt soll es hier für $A^{(i)} = \bigotimes_{j<i} \mathbb{1} \otimes A \otimes \bigotimes_{j>i} \mathbb{1}$ stehen.

Genauso ist $|\varphi\rangle^{(i)} = |\varphi^{(i)}\rangle = \varphi^{(i)}$ als Projektion auf den durch i bezeichneten Untervektorraum definiert.

Falls die Dimension von A größer als 2 ist, legt diese Notation die Abbildung nicht genau fest, es ist aber immer durch den Kontext eine kanonische Zuordnung gegeben. Insbesondere die out-Funktion (Abschnitt 4) stellt einen Zusammenhang zwischen dem Raum, in dem der Operator definiert und dem, in dem er angewendet wird, dar.

2.1.1 Bloch-Sphäre und Pauli-Gruppe

Die Pauli-Gruppe besteht aus $\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$, $\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ mit der Eigenschaft $\sigma_i = i\sigma_j\sigma_k$, i,j,k Permutation aus x,y,z. Die Operatoren haben die Eigenwerte ± 1 und die Eigenvektoren $|0\rangle_x = |+\rangle = (|0\rangle + |1\rangle)$, $|1\rangle_x = |-\rangle = (|0\rangle - |1\rangle)$, $|0\rangle_y = (|0\rangle + i|1\rangle)$, $|1\rangle_y = (|0\rangle - i|1\rangle)$, $|0\rangle_z = |0\rangle_{Rechenbasis} = |0\rangle$ und $|1\rangle_z = |1\rangle_{Rechenbasis} = |1\rangle$.

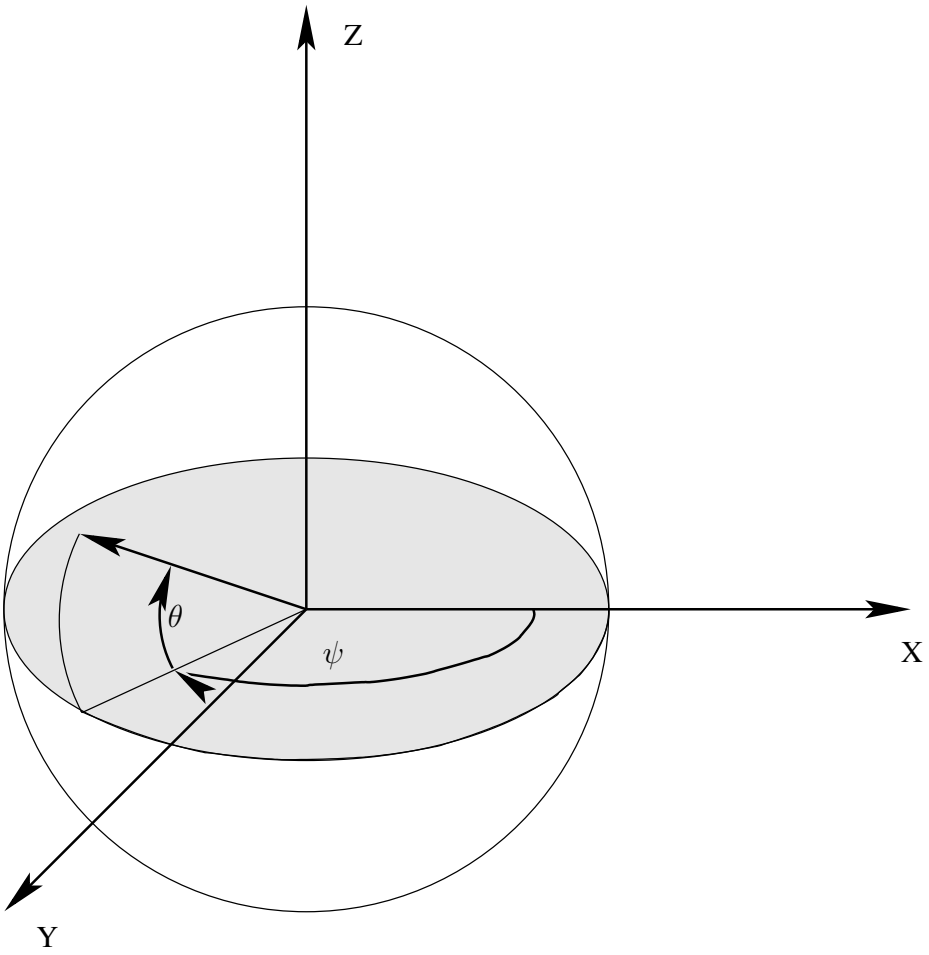


Abbildung 2.1: Bloch Sphäre

Der Zustand eines Zweiniveausystems, auch Qubit[2] genannt, ist ein Vektor $\begin{pmatrix} \alpha \\ \beta \end{pmatrix} \in \mathbb{C}^2$ mit $|\alpha|^2 + |\beta|^2 = 1$, das sich damit auch so $e^{i\zeta} \begin{pmatrix} cos\theta/2 \\ e^{i\gamma} sin\theta/2 \end{pmatrix}$ schreiben lässt, mit $\theta, \gamma, \zeta \in [0, 2\pi[$. Wenn man die globale Phase ζ, die physikalisch irrelevant ist, weglässt, kann man den Vektor als einen Punkt auf der Einheitskugel $\{x \in \mathbb{R}^3 | \, |x| = 1\}$ im $\mathbb{R}^3$ darstellen.

Die Elemente der Pauli-Gruppe sind dann Drehungen des Zustandsvektors um die x, y und z-Achse. Die unitären Transformationen auf den reinen Zuständen eines Quantensystems lassen sich also als Rotationen eines Körpers verstehen.

[2] nach Schumacher, 1995. Quelle: [9], S.58.

2.2 Quantenturingmaschinen

Eine (Einband-)Quantenturingmaschine mit einer endlichen Zustandsmenge Q und einem endlichen Alphabet Σ wird durch ein 4-Tupel $(Q, \Sigma, \delta, q_0, Q_{halt})$ dargestellt, wobei q_0 der Anfangszustand, Q_{halt} die Menge der Endzustände und $\delta : \Sigma \times Q \times \Sigma \times Q \times \{\leftarrow, \odot, \rightarrow\} \longrightarrow \{x \in \mathbb{C} | \, |x| \leq 1\}$ die Übergangsfunktion, die jedem Übergang von $\Sigma \times Q$ nach $\Sigma \times Q$ mit eventueller Bewegung des Lesekopfes einekomplexe Amplitude zuweist, sind. Betrachtet man nun ein vollständiges Baumdiagramm des zeitlichen Ablaufs einer Quantenturingmaschine (Abbildung 2.2), dann kann man jedem Übergang den entsprechenden Wert der δ-Funktion und damit jeder Konfiguration eine komplexe Amplitude als Summe über ihre Vorgängerkonfigurationen multipliziert mit der jeweiligen Übergangswahrscheinlichkeit zuordnen.

Genauer: Sei $c = (t, h, q)$, wobei $t : \mathbb{Z} \rightarrow \Sigma \cup \{B\}$ - mit B einem Symbol für eine leere Bandposition ("blank") - der Bandinhalt, $h \in Z$ die Kopfposition und $q \in Q$ der Zustand der Kontrolle ist, eine Basiskonfiguration. Dann gibt es über dem Basiskonfigurationsraum C_M der QTM M eine Abbildung $\delta^* : C_M \rightarrow C_M$, mit $\delta^*((t,h,q),(t',h',q')) = \begin{cases} 0, \text{ falls } (t(h),q) \text{ nicht im Definitionsbereich von } \delta \\ \delta(t(h),q), \text{ sonst} \end{cases}$ Dieser Basiskonfigurationsraum induziert nun den Konfigurationsraum C_M^α der QTM M aus Überlagerungen der Basiskonfigurationen c. Also $C_M^\alpha = C_M \rightarrow \{x \in \mathbb{C} | \, |x| \leq 1\}$. Ein Element $(\alpha)_c$ aus C_M^α lässt sich als (abzählbar unendlich langer, da C_M abzählbar unendlich ist) Vektor schreiben. Damit lassen sich die Übergänge im Konfigurationsraum der C_M durch eine Matrix E_M darstellen, die diese lineare Abbildung für alle α_c repräsentiert.

$$\begin{pmatrix} \alpha_{c_0} \\ \alpha_{c_1} \\ \alpha_{c_2} \\ \alpha_{c_3} \\ \vdots \end{pmatrix} = \begin{pmatrix} \delta^*(c_0,c_0') & \delta^*(c_0,c_1') & \delta^*(c_0,c_2') & \delta^*(c_0,c_3') & \cdots \\ \delta^*(c_1,c_0') & \delta^*(c_1,c_1') & \delta^*(c_1,c_2') & \delta^*(c_1,c_3') & \\ \delta^*(c_2,c_0') & \delta^*(c_2,c_1') & \delta^*(c_2,c_2') & \delta^*(c_2,c_3') & \\ \vdots & & & & \ddots \end{pmatrix} \begin{pmatrix} \alpha_{c_0'} \\ \alpha_{c_1'} \\ \alpha_{c_2'} \\ \alpha_{c_3'} \\ \vdots \end{pmatrix}$$

Die Matrix ist also wie die Konfigurationsvektoren unendlich.

Damit das 5-Tupel wirklich eine QTM realisiert, muss die Funktion unitär sein. Das schliesst ein, dass die lokale Wahrscheinlichkeitsbedingung $\Sigma_c \left|\alpha_c\right|^2 = 1$ durch die δ-Übergangsfunktion erfüllt sein muss.

Mit der Matrix lassen sich nun alle α_c für jeden Zeitschritt der Entwicklung der QTM berechnen: Man hat im Zeitschritt 0 für die Ausgangskonfiguration $(\lambda x.B, 0, q_0)$ die Amplitude 1, für alle anderen die Amplitude 0. Dann wendet man die Matrix auf diesen Konfigurationsvektor an und erhält die Amplituden für die QTM im Zeitschritt 1 usw.

Das Ergebnis dieses Rechenprozesses misst man nach n Schritten. Man erhält dann die Basiskonfiguration c_i mit der Wahrscheinlichkeit $P(c_i) = |(E_M^n(\alpha)_{\text{Start}})_{c_i}|^2$.

2.3 Quantennetzwerke

Der Zusammenhang zwischen Quantenturingmaschinen und der zugrundeliegenden Darstellung der physikalischen Verhältnisse ist leider komplex und nicht einfach zu durch-

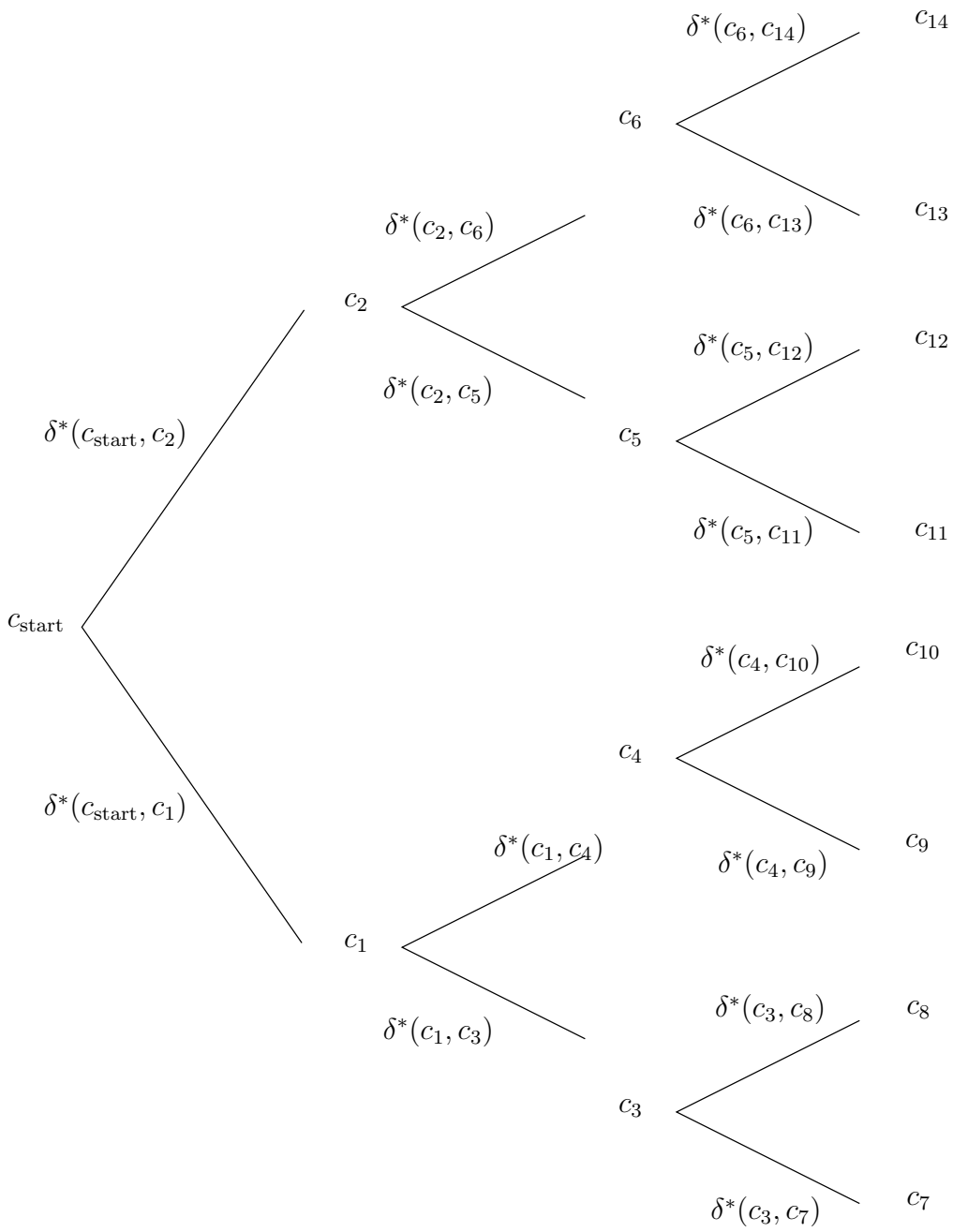


Abbildung 2.2: Berechnungsbaum einer Quantenturingmaschine

schauen. Eine andere Möglichkeit informationsverarbeitende Prozesse darzustellen, ist die Verwendung von Netzwerken aus elementaren Gattern. Diese für die klassische Berechenbarkeits- und Komplexitätstheorie als äquivalent bekannte Methode ist bei der Quanteninformationsverarbeitung normalerweise den QTMs vorzuziehen, da durch den Aufbau mit Gattern, die elementaren physikalischen Realisierungen entsprechen, die Struktur des Berechnungsmodells luzider wird.

Die Gleichwertigkeit der Informationsverarbeitung von Quantenturingmaschinen und Quantennetzwerken wurde von Yao [13] gezeigt. Im Rest dieser Abhandlung werden die Sachzusammenhänge nur noch mit Hilfe von Netzwerken erläutert.

2.3.1 Darstellung von Netzwerken

Quantennetzwerke werden als endliche, zyklenfreie, gerichtete Graphen (V, K, t) mit V der Knotenmenge (den "Gattern") und $K \subset ((V \times \mathbb{N}) \cup \{B\}) \times ((V \times \mathbb{N}) \cup \{B\})$ (wobei links ein Paar (Startknoten, "Gatterausgang") und rechts ein Paar (Zielknoten, "Gattereingang") oder "B" für einen offenen Ein- bzw. Ausgang steht) der Kantenmenge (den "Kabeln") dargestellt. Die "Gatterausgänge" müssen wie die "Gattereingänge" für alle Kanten eines Knotens verschieden sein: $\forall(p_{\text{in}}, p_{\text{out}}) \neq (p'_{\text{in}}, p'_{\text{out}}) \in K.p_{\text{in}} \neq p'_{\text{in}} \wedge p_{\text{out}} \neq p'_{\text{out}} \vee B \in \{p_{\text{in}}, p_{\text{out}}, p'_{\text{in}}, p'_{\text{out}}\}$. Die Knoten - die mit der Transformation beschriftet sind, die sie ausführen $(t : V \to \{U | U \text{ Transformation}\})$ - müssen heraus- wie hereinführende Kanten die gleichen "Gatterein-" bzw. "ausgänge" des Gatters belegen $(\forall v \in V.\pi_{\text{right}}\pi_{\text{left}}((\{v\} \times \mathbb{N}) \times (V \times \mathbb{N}) \cap K) = \pi_{\text{right}}\pi_{\text{right}}((V \times \mathbb{N}) \times (\{v\} \times \mathbb{N}) \cap K))$.

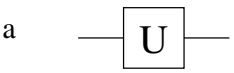
Durch die Kanten (als "Quantenkabel") fliesst der Zustand jeweils eines Qubits und an den Knoten wird die bezeichnete Operation ausgeführt.

Üblicherweise werden die Graphen so gezeichnet, dass ein informationsverarbeitender Prozess als eine zeitliche Abfolge der Anwendung der Tranformationen an den Knoten von links nach rechts erscheint.

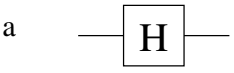
Nun zur graphischen Darstellung einzelner Gatter, aus denen die Netzwerke bestehen.

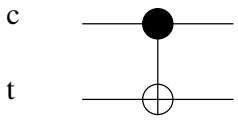
2.3.1.1 Die Gatter

Das Gatter für eine unitäre **1-Qubit-Rotation** U in der von Feynman eingeführten Darstellungsweise. Es repräsentiert $|a_{\text{out}}\rangle = U\,|a_{\text{in}}\rangle$.

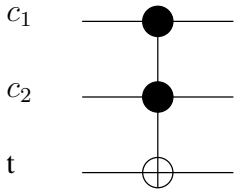


Ein spezielles Gatter dieser Form ist das **Hadamard** mit der Gleichung $|a_{\text{out}}\rangle = (\alpha + \beta)\,|0\rangle + (\alpha - \beta)\,|1\rangle = H(\alpha\,|0\rangle + \beta\,|1\rangle) = H\,|a_{\text{in}}\rangle$.

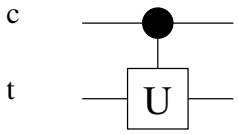




Von den Gattern auf zwei Qubits ist das bedeutendste das **CNOT**, das einen σ_x-Spinflip auf dem Target-Qubit t macht, falls das Kontrollqubit c gleich 1 ist und sonst t durchschleift. Das Kontrollqubit bleibt immer unverändert. Dieses Gatter hat eine spezielle Darstellung: Ein Strich, wobei am Schnittpunkt mit der Leitung des Kontrollqubit ein voller Punkt, am Schnittpunkt mit der Leitung des Target-Qubits ein gekreuzter Punkt gemalt wird.
Es führt $|c_{\text{out}}, t_{\text{out}}\rangle = \text{CNOT}\,|c_{\text{in}}, t_{\text{in}}\rangle = (|0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes \sigma_x)\,|c_{\text{in}}, t_{\text{in}}\rangle$ aus.



Das **Toffoli** heisst auch CCNOT-Gatter: beide durch volle Punkte markierten Kontrollleitungen c_1 und c_2 müssen auf 1 sein, damit das Target-Qubit t geflippt wird. Formel: $|c_{1\text{out}}, c_{2\text{out}}, t_{\text{out}}\rangle = \text{CNOT}\,|c_{1\text{in}}, c_{2\text{in}}, t_{\text{in}}\rangle = (\bigotimes_{i=1}^{3}\mathbb{1} + |11\rangle\langle 11| \otimes (\sigma_x - \mathbb{1}))\,|c_{1\text{in}}, c_{2\text{in}}, t_{\text{in}}\rangle$



Das **kontrollierte Gatter** ist die eine Verallgemeinerung des CNOT und hat die Funktion: $|c_{\text{out}}, t_{\text{out}}\rangle = \text{CNOT}\,|c_{\text{in}}, t_{\text{in}}\rangle = (|0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes U)\,|c_{\text{in}}, t_{\text{in}}\rangle$, wendet also die untäre Transformation auf das Target-Qubit an, falls das Kontrollqubit 1 ist und entspricht sonst der Identität.
Das läßt sich natürlich weiter verallgemeinern und in [1] wird für ein kontrolliertes Gatter mit n Kontrollqubits $\wedge_n(U)$ geschrieben und es eben mit n vollen Punkten gezeichnet. Ich brauche solche Gatter hier aber nicht.

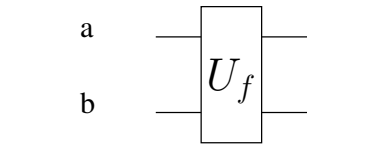
2.3.1.2 Zusammenschaltungen

Beim Zusammenschalten von Quantennetzwerken verbindet man Eingangsknoten des einen Netzwerks mit Ausgangsknoten des anderen, immer eins-zu-eins, und markiert die resultierenden Transformationsknoten mit der Identität $\mathbb{1}$. Seien also (V, K, t) und (V', K', t') zwei Quantennetzwerke mit disjunkter Knotenmenge $V \cap V' = \emptyset$ und j eine injektive Funktion von $J \subset \{v \in V | (\{v\} \times \mathbb{N}) \times \{B\} \in K\}$ nach $V'_{\text{in}} = \{v \in V | \{B\} \times (\{v\} \times \mathbb{N}) \in K\}$ dann ist $(V \cup V', K \cup K' - (dom(j) \cup range(j)) \cup \{((\pi_{\text{left}} v, \pi_{\text{right}} j(v)) | v \in J\}, t \cup t')$ der Graph der Hintereinanderschaltung.

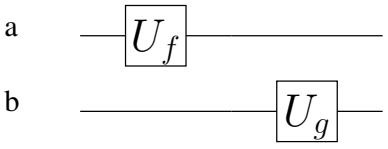
Gezeichnet werden sie einfach hintereinander, evtl. verlegt man noch die Ein- bzw.

Ausgangsknoten in eine Höhe.

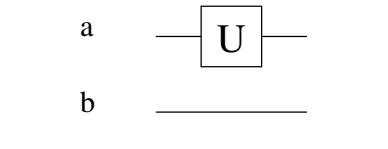
$\mathbb{1} \otimes A$ $\qquad A^{(b)}$



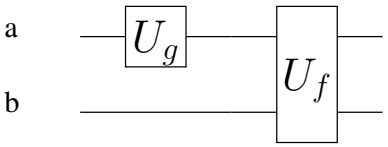
U_f $\qquad U_f^{(a,b)}$



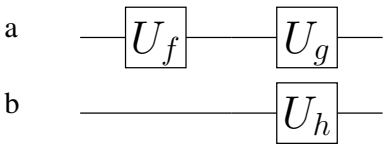
$(U_f \otimes \mathbb{1})(\mathbb{1} \otimes U_g)$ $\qquad U_f^{(a)} U_g^{(b)}$



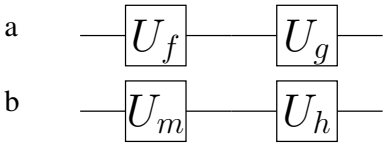
$U \otimes \mathbb{1}$ $\qquad U^{(a)}$



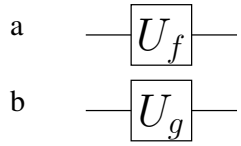
$(U_g \otimes \mathbb{1})U_f$ $\qquad U_g^{(a)} U_f^{(a,b)}$



$$\begin{aligned} & (U_f \otimes \mathbb{1})(U_g \otimes U_h) \\ = \; & U_f U_g \otimes U_h \end{aligned} \qquad U_f^{(a)} U_g^{(a)} U_h^{(b)}$$



$$\begin{aligned} & (U_f \otimes U_m)(U_g \otimes U_h) \\ = \; & U_f U_g \otimes U_m U_h \end{aligned} \qquad \begin{aligned} & U_f^{(a)} U_g^{(a)} U_m^{(b)} U_h^{(b)} \\ = \; & U_m^{(b)} U_h^{(b)} U_f^{(a)} U_g^{(a)} \end{aligned}$$



$$U_f \otimes U_g$$

$$\begin{aligned} & U_f^{(a)} U_g^{(b)} \\ = \; & U_g^{(b)} U_f^{(a)} \end{aligned}$$

In [1] wird ein allgemeines Verfahren zur Umwandlung von unitären Transformationen auf n Qubits auf ein Netzwerk aus 1- und 2-Qubit-Gattern dargelegt und für eine ganze Reihe von Gattern exemplifiziert. Meine Implementierung des Toffolis ist nach dieser Methode gestrickt.

Das Umdrehen - also Eingänge vertauschen, die Richtung der Graphenpfeile verkehren und die Transformationen A an den Knoten in $A^\dagger$ ändern - eines Qantennetzwerkes für U erzeugt eines für $U^\dagger$. Präzise: Sei (V, K, t) ein Quantennetzwerk für U. Dann ist $V_{\text{trans}} = \{v \in V || (\{v\} \times \mathbb{N}) \times (V \times \mathbb{N}) \cap K| = |(V \times \mathbb{N}) \times (\{v\} \times \mathbb{N}) \cap K|\}$. Mit $K^\dagger = \{(p, p') | (p', p) \in K\}$ und $t^\dagger : V_{\text{trans}} \to \{U | U \text{ Transformation}\}$, $t^\dagger(v) = t(v)^\dagger$ ist $(V, K^\dagger, t^\dagger)$ das umgedrehte Quantennetzwerk.

2.3.2 Arithmetische Grundlagen für die hier vorgestellten Netzwerke

Da das Ziel dieser Arbeit ist, ein Messmuster für ein Netzwerk zu entwerfen, das eine Exponentiation modulo N berechnet, werden hier Berechnungsverfahren für die Rechenarten Exponenzierung, Multiplikation und Addition im binären Zahlensystem dargestellt, die dann bei der Konstruktion der Schaltkreise Verwendung finden.

2.3.2.1 v.l.n.r.-Exponentiation ("ägyptisch")

Man startet mit der 1 im Register. Dann führt man für jedes Bit des Exponenten von links nach rechts, also vom höchstwertigsten zum niederwertigsten Bit, folgendes aus:

1. Quadriere das Register
2. Multipliziere es mit a, falls das Exponentenbit gleich 1, sonst tue nichts.

In Formeln

$A_0 = 1$, $A_{i+1} = A_i^2 a^{b_{n-i}}$

mit n der Bitlänge von b, also $(b)_{\text{dual}} = b_{n-1} b_{n-2} b_{n-3} \dots b_0 = \sum_{i=1}^{n} 2^{n-i} b_{n-i}$.

Damit ist $A_n = (((\cdots (1 a^{b_{n-1}})^2 a^{b_{n-2}})^2 a^{b_{n-3}})^2 \dots)^2 a^{b_0} =$

$= \prod_{i=1}^{n} a^{2^{n-i} b_{n-i}} =$, da $(xy)^2 = x^2 y^2$

$= a^{\sum_{i=1}^{n} 2^{n-i} b_{n-i}} = a^b$

2.3.2.2 v.r.n.l.-Exponentiation

Start mit 1 im Ergebnisregister. Für jedes Bit des Exponenten von rechts nach links, also vom niederwertigsten zum höchstwertigsten Bit, wird folgendes ausgeführt

1. Multipliziere das Eingaberegister mit dem Ergebnisregister, falls das Exponentenbit gesetzt ist.

2. Quadriere das Eingaberegister.

$A_0 = 1$, $A_{i+1} = A_i a^{2^i b_i}$

Damit ist $A_n = (((\cdots(1a^{2^0 b_{n-1}})a^{2^1 b_{n-2}})a^{2^2 b_{n-3}})\ldots)a^{2^{n-1} b_{n-1}} =$

$= \prod_{i=0}^{n-1} a^{2^i b_i} =$

$= a^{\sum_{i=0}^{n-1} 2^i b_i} = a^b$

Der Vorteil der v.r.n.l.-Exponentiation ist, dass eine Vorausberechnung der Potenzen von a möglich ist (da a im Shor-Algorithmus eine feste gegebene Zahl ist) und dadurch der entsprechende Schaltkreis kleiner wird. Dem steht der Vorteil des v.l.n.r.-Exponentierers gegenüber, sehr regelmäßige Strukturen zu haben, die sich eventuell besser verstehen und experimentell beherrschen lassen.

2.3.2.3 v.l.n.r.-Multiplikation

Man startet mit der 0 im Register. Dann führt man für jedes Bit des 2.Faktors von links nach rechts, also vom höchstwertigsten zum niederwertigsten Bit, folgendes aus:

1. Multipliziere das Register mit 2

2. Addiere a, falls das entsprechende Bit des 2. Faktors gleich 1, sonst tue nichts.

In Formeln

$A_0 = 0$, $A_{i+1} = 2A_i + ab_{n-i}$

Damit ist $A_n = (((\cdots(0 + ab_{n-1}) * 2 + ab_{n-2}) * 2 + ab_{n-3}) * 2\ldots) * 2 + ab_0 =$

ausmultipliziert$= 2^{n-1}ab_{n-1} + 2^{n-2}ab_{n-2} + \cdots + 2^0 ab_0 =$

$= a(2^{n-1}b_{n-1} + 2^{n-2}b_{n-2} + \cdots + 2^0 b_0) =$

$= \sum_{i=1}^{n} 2^{n-i} b_{n-i} = ab$

2.3.2.4 v.r.n.l.-Multiplikation (Schulbuchmultiplikation, "russian peasant")

Man startet mit der 0 im Ergebnisregister. Dann führt man für jedes Bit des 2.Faktors von rechts nach links, also vom niederwertigsten zum höchstwertigsten Bit, folgendes aus:

1. Addiere das Eingaberegister zum Ergebnisregister, falls das Bit des 2. Faktors gleich 1, sonst tue nichts.

2. Multipliziere das Eingaberegister mit 2

Damit

$A_0 = 0$, $A_{i+1} = A_i + 2^i ab_i$

Damit ist $A_n = (((\cdots(0 + 2^0 ab_0) + 2^1 ab_1) + 2^2 ab_2) + \ldots) + 2^{n-1} ab_{n-1} =$

$= a(2^0 b_0 + 2^1 b_1 + 2^2 b_2 \cdots + 2^{n-1} b_{n-1}) =$

$= \sum_{i=1}^{n} 2^{n-i} b_{n-i} = ab$

				a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	b	
$2^0 a$				1	0	1	0	0	1	0	0	1	b_0
$2^1 a$			1	0	1	0	0	1	0	0		0	b_1
$2^2 a$		1	0	1	0	0	1	0	0			1	b_2
	1	0	0	0	1	1	1	0	1	0	0		

Bei der Multiplikation besteht kein wesentlicher Unterschied zwischen v.l.n.r. und v.r.n.l. Entweder wird das Eingaberegister geschoben und dann addiert, oder das Ergebnisregister wird geschoben und a hinzugezählt.

2.3.2.5 modulo Arithmetik

Für den Shor-Algorithmus benötigt man eine Funktion $a^b \bmod N$, also modulo-Arithmetik. Notation: Der Ausdruck "mod" wird auf zwei Weisen gebraucht. Entweder wie hier als Funktion $x \bmod N : \mathbb{N} \times (\mathbb{N} - \{0\}) \to \{0, \ldots, N-1\}$ mit der (rekursiven) Definition $x \bmod N = \begin{cases} x, & \text{falls } 0 \leq x < N \\ x - N \bmod N, & \text{sonst} \end{cases}$, die den Rest der ganzzahligen Division von x durch N zurückgibt oder als Bezeichnung einer Kongruenzrelation $a \equiv b \bmod N$ mit $a \equiv b \bmod N \Leftrightarrow a \bmod N = b \bmod N$.

Es gilt:

1. $a + b \bmod N = (a \bmod N) + b \bmod N$, da mit $a = kN + r$ gilt $a + b \bmod N = (kN + r) + b \bmod N = r + b \bmod N = ((a \bmod N) + b) \bmod N$. Da $+$ kommutativ ist, gilt auch $a + b \bmod N = (a + (b \bmod N)) \bmod N$.

2. $ab \bmod N = (a \bmod N)b \bmod N$, da $ab \bmod N = (\sum_{i=1}^{b} a) \bmod N =^{\text{mit 1.}} \sum_{i=1}^{b} (a \bmod N) \bmod N = (a \bmod N)b \bmod N$. Da die Multiplikation kommutativ ist, gilt auch $ab \bmod N = (a(b \bmod N)) \bmod N$.

3. $a^b \bmod N = (a \bmod N)^b \bmod N$, da $a^b \bmod N = (\prod_{i=1}^{b} a) \bmod N =^{\text{mit 2.}} \prod_{i=1}^{b} (a \bmod N) \bmod N = (a \bmod N)^b \bmod N$.

Damit lassen sich die oben angegebenen Algorithmen direkt zu solchen für modulare Arithmetik umwandeln, z.B. ist $A_0 = 0$, $A_{i+1} = 2A_i \bmod N + ab_{n-i} \bmod N$ ein v.l.n.r.-Multiplizierer modulo N. Wichtig für die Implementierung ist noch zu sehen, dass mit $a < N$ alle vorkommenden Zahlen $< 2N$ sind.

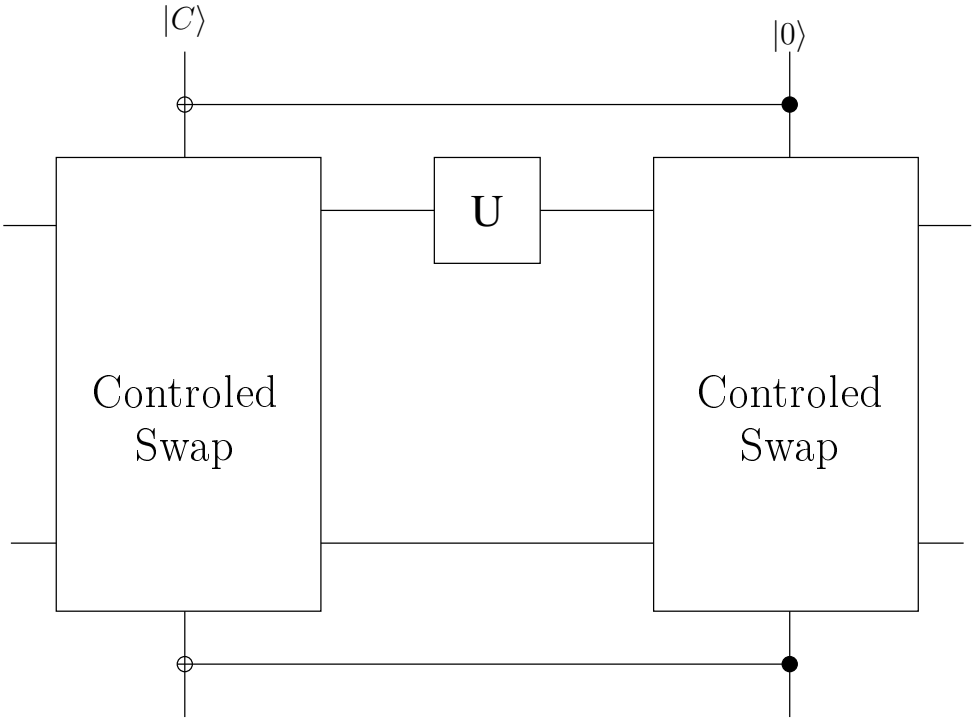
Das Berechnen von $a \bmod N$ mit $0 \leq a < 2N$ erfolgt durch Subtraktion von N von a (was der Addition des 2er-Komplements von N zu a entspricht), falls dabei ein Übertrag auftritt, sonst ist es einfach a.

Hässlicherweise ist - obwohl die Funktion $(a, b) \mapsto (a, a + b \bmod N)$ bijektiv auf $\{0, ..., N-1\}^2$ ist - die kanonische Übersetzung dieser Funktion ins Binärsystem nicht reversibel, da für Fälle, in denen N keine Zweierpotenz ist, problematische Paare wie $(0, 0)$ und $(0, N)$ auftreten können. Das führt in meiner Implementierung dazu, dass zusätzlich Hilfsqubits benötigt werden, die sich nur mit viel Aufwand - invertieren der kompletten Funktion - wieder zurücksetzen lassen.

2.3.3 Arithmetische Netzwerke

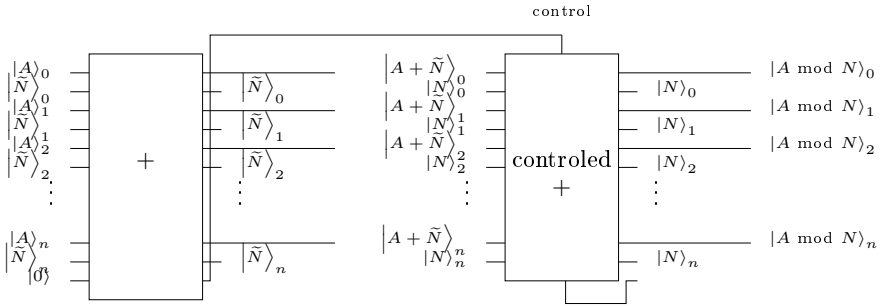
2.3.3.1 Controled U

Zur Implementierung der Multiplikation und Exponentiation braucht man eine bedingte Addition. Ein generelles Schema für bedingte Operationen ist



aus kontrollierten Swaps. Die Hilfsqubits müssen wieder entschränkt werden, was durch Umkehren dieses Gatters am besten am Ende der Gesamtoperation geschieht.

2.3.3.2 modulo N

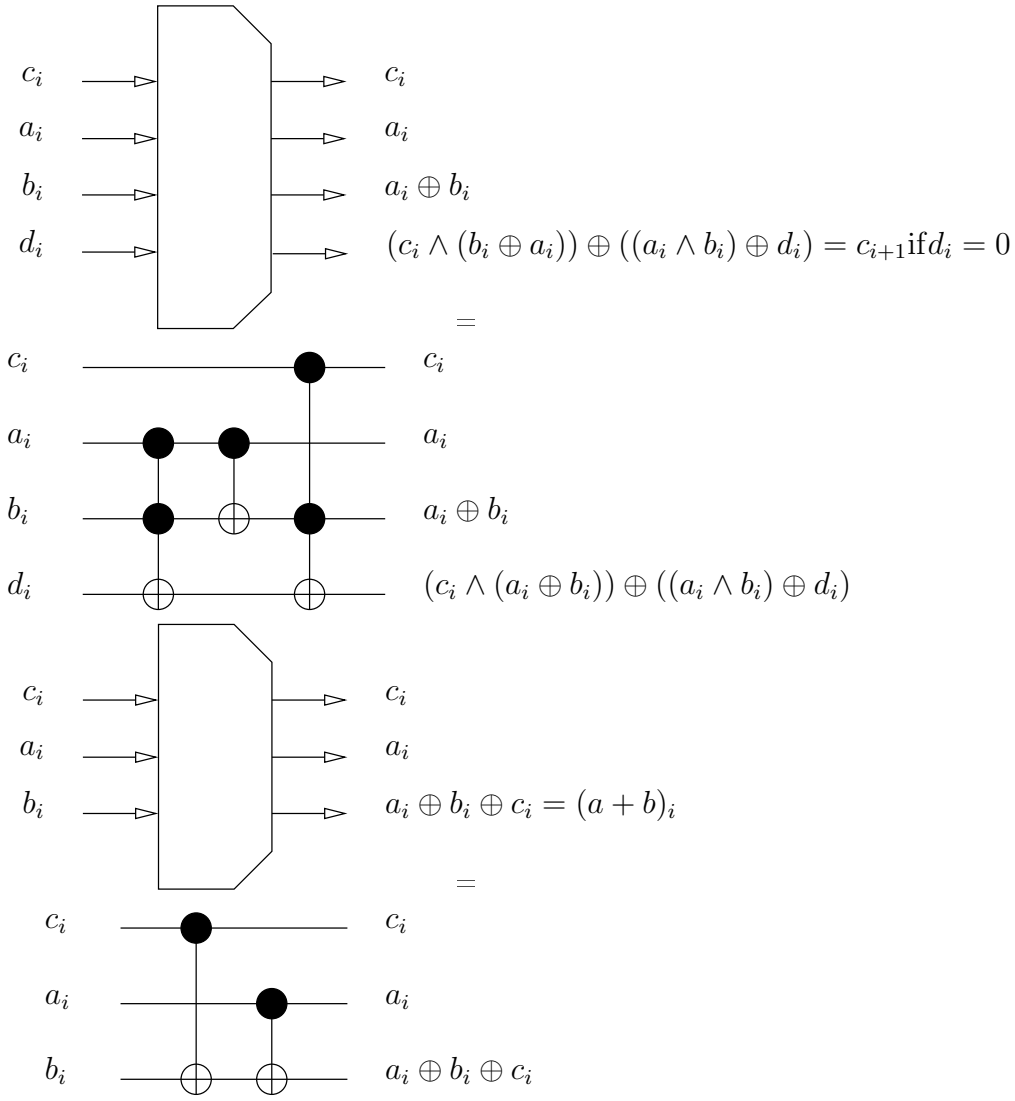


Die Berechnung der modulo-Funktion lässt zwingendermaßen ein verschränktes Hilfsqubit entstehen, wie oben (2.3.2.5) bereits angemerkt. Auch hier kann die Entschränkung durch Umkehrung des Schaltkreises erfolgen.[3]

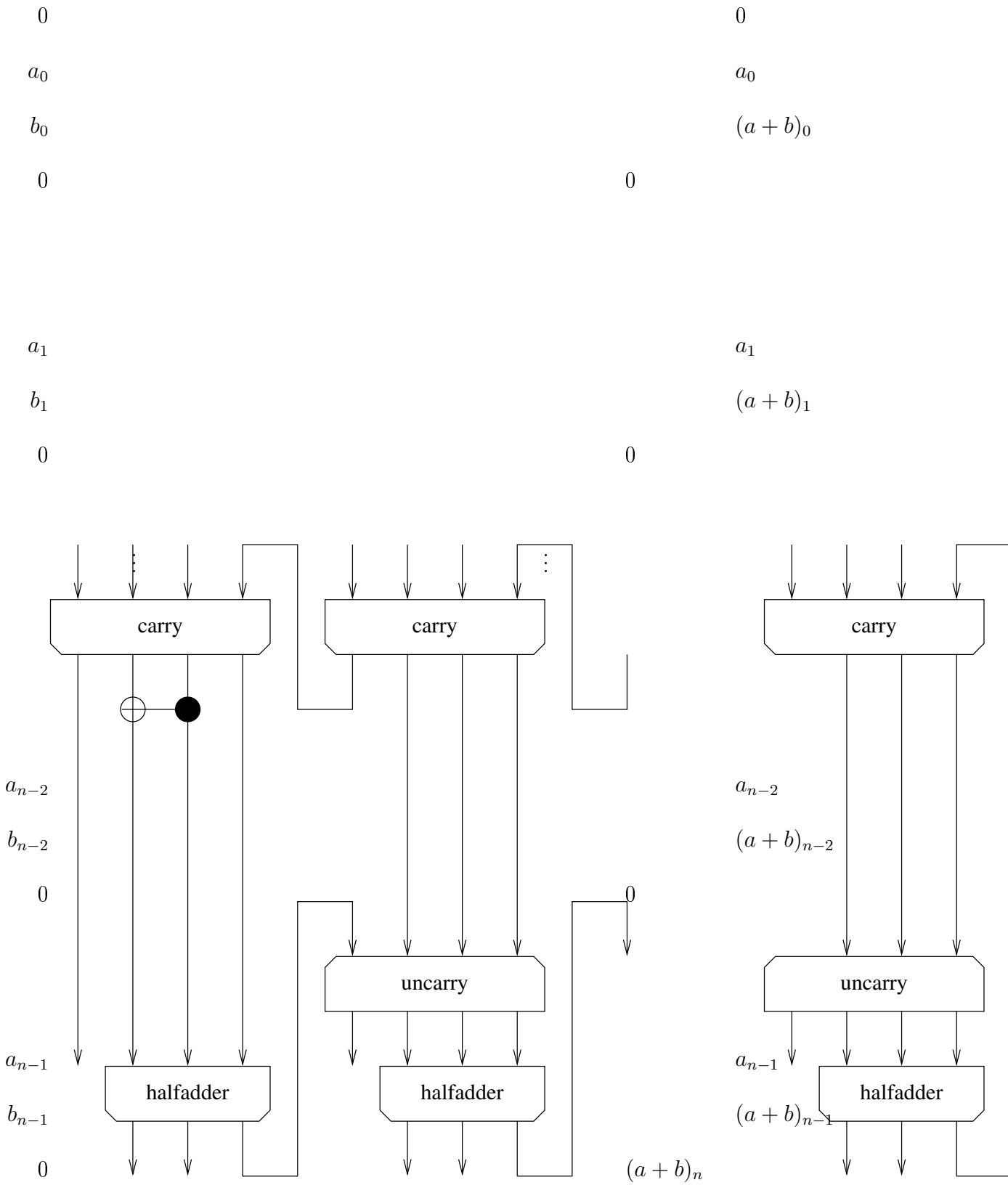
[3] ein falscher Schaltkreis dazu auch z.B. in [9], S. 97

Abbildung 2.3: Addierer nach Gruska

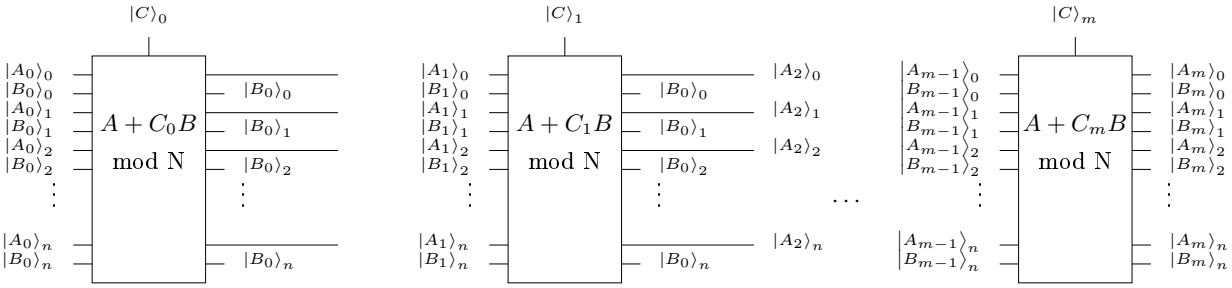
2.3.3.3 Addierer nach Gruska



Die bildliche Verkehrung eines Gatters bedeutet die Anwendung der inversen Transformation, hier verwendet zur Rücksetzung der Carry-Hilfsqubits nach dem Schema (oder Trick) von Bennett ([2]).

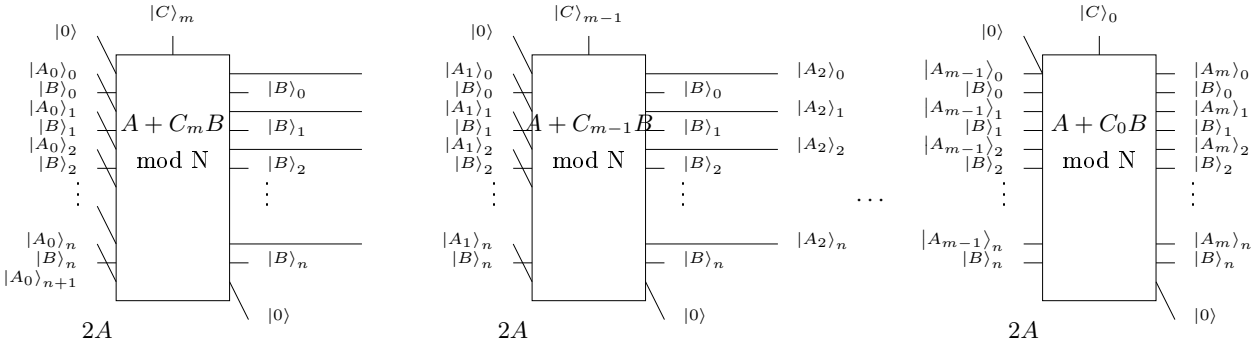

0
a_0
b_0
0
0
a_0
$(a+b)_0$
a_1
b_1
0
a_1
$(a+b)_1$
carry
carry
carry
a_{n-2}
b_{n-2}
0
a_{n-2}
$(a+b)_{n-2}$
0
uncarry
uncarry
a_{n-1}
b_{n-1}
0
halfadder
halfadder
a_{n-1}
$(a+b)_{n-1}$
halfadder
$(a+b)_n$

2.3.3.4 v.r.n.l.-Multiplikation



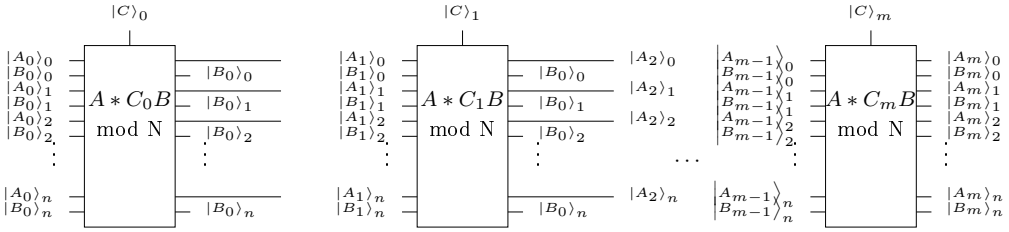
mit $B_i = 2^i a \bmod N$, A_i dem Akkumulatorregister ($A_0 = 0$) und $C = b$ dem 2. Faktor.

2.3.3.5 v.l.n.r.-Multiplikation



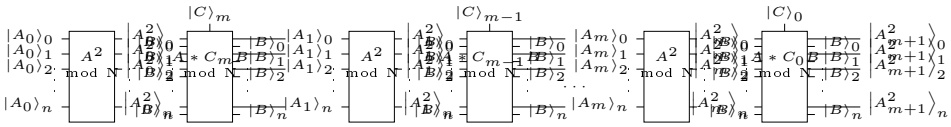
mit $B = a \bmod N$, A_i dem Akkumulatorregister ($A_0 = 0$) und $C = b$ dem 2. Faktor.

2.3.3.6 v.r.n.l.-Exponentiation



Mit $A_0 = 1$ und $B_i = a^{2^i} mod N$ ist $A_m = a^C$.

2.3.3.7 v.l.n.r.-Exponentiation



Mit $A_0 = 1$ und $A_{i+1} = A_i^2 * B^{C_{m-i}}$ erhält man $A_{m+1} = B^C$ (jeweils modulo N).

2.3.4 QFT-Netzwerk

Die diskrete Fouriertransformation bildet komplexwertige diskrete Funktionen $f : \{0, ..., q-1\} \to \mathbb{C}$ in den Phasenraum ab: $f^*(n) = \sum_{j=0}^{q-1} e^{2\pi i n j/q} f(j)$.

Die Quantenfouriertransformation ist eine unitäre Abbildung, die eine durch die komplexwertigen Amplituden der Basiszustände repräsentierte Funktion f von $\{0, ..., q-1\}$ nach $\mathbb{C}$ in ihre diskrete Fouriertransformierte umwandelt. Für einen Hilbertraum der Größe $q = 2^n$ ist also die $\mathrm{QFT}_q : |a\rangle \mapsto 1/\sqrt{q} \sum_{j=0}^{q-1} e^{2\pi i a j/q} |j\rangle$, als Matrix geschrieben

$$\mathrm{QFT}_q = \frac{1}{\sqrt{q}} \begin{pmatrix} r^0 & r^0 & r^0 & r^0 & \dots & r^0 \\ r^0 & r^1 & r^2 & r^3 & \dots & r^{(q-1)} \\ r^0 & r^2 & r^4 & r^6 & \dots & r^{2(q-1)} \\ r^0 & r^3 & r^6 & r^9 & \dots & r^{3(q-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ r^0 & r^{(q-1)} & r^{2(q-1)} & r^{3(q-1)} & \dots & r^{(q-1)^2} \end{pmatrix}$$

mit $r = e^{2\pi i/q}$. Damit gilt: $QFT_q \sum_{j=0}^{q-1} f(q) |q\rangle \to \sum_{j=0}^{q-1} f^*(q) |q\rangle$, es wird also, wie oben schon geschrieben, die jedem Quantenzustand zugeordnete Funktion von den Basiszuständen zu den entsprechenden Amplituden fouriertransformiert.

Die Zahl der Multiplikationen lässt sich bei der diskreten Fouriertransformation durch Berechnung von Teilprodukten, die in mehreren Ergebnissen vorkommen, erheblich von $O(q^2)$ auf $O(q \log q)$ reduzieren (schnelle Fouriertransformation).

Das ist allerdings immer noch exponentiell mehr als bei der Quantenfouriertransformation, die mit nur $O((\log q)^2) = O(n^2)$ elementaren unitären Transformationen auskommt. Hier zeigt sich der Quantenparallelismus, denn es sind ja q verschiedene Funktionswerte als Eingabe, von denen (da die Abbildung bijektiv ist) auch alle etwas zum Ergebnis beitragen, man also klassisch nicht schneller als linear sein kann.

Die Zerlegung in Elementartransformationen sieht folgendermaßen aus (Indizes an Variablen bedeuten das indizierte Bit der Binärdarstellung des Wertes der Variablen, also $(a)_{\mathrm{dual}} = a_{n-1}a_{n-2}a_{n-3}\dots a_o$ und für die beiden patologischen Fälle einer endlichen Folge aus den Ziffern von a soll gelten: $a_{n-1}...a_n := 0$ und $a_0...a_0 := a_0$):

Der Zustand

$$\frac{1}{\sqrt{q}} \bigotimes_{k=0}^{n-1} (|0\rangle + \prod_{j=0}^{k} e^{2\pi i/2^{j+1} a_{k-j}} |1\rangle)$$

ist die QFT des Zustandes $|a\rangle$, denn

$$\begin{aligned} = \quad & \frac{1}{\sqrt{q}} (|0\rangle + e^{2\pi i/2^1 a_0} |1\rangle) \otimes (|0\rangle + e^{2\pi i/2^1 a_1} e^{2\pi i/2^2 a_0} |1\rangle) \\ & \otimes (|0\rangle + e^{2\pi i/2^1 a_2} e^{2\pi i/2^2 a_1} e^{2\pi i/2^3 a_0} |1\rangle) \otimes \dots \otimes (|0\rangle + e^{2\pi i/2^1 a_{n-1}} \dots e^{2\pi i/2^n a_0} |1\rangle) \end{aligned}$$

$$= \frac{1}{\sqrt{q}}(|0\rangle + e^{2\pi i 0,a_0}|1\rangle) \otimes (|0\rangle + e^{2\pi i 0,a_1 a_0}|1\rangle)$$
$$\otimes(|0\rangle + e^{2\pi i 0,a_2 a_1 a_0}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2\pi i 0,a_{n-1}\ldots a_0}|1\rangle)$$

$$= \frac{1}{\sqrt{q}} \sum_{s=0}^{q} \prod_{j=0}^{n-1} (e^{2\pi i 0,a_j \ldots a_0})^{s_{n-1-j}} |s\rangle$$

$$= \frac{1}{\sqrt{q}} \sum_{s=0}^{q} \prod_{j=0}^{n-1} e^{2\pi i 0,a_{n-1-j}\ldots a_0 s_j} |s\rangle$$

$$= \frac{1}{\sqrt{q}} \sum_{s=0}^{q} e^{2\pi i \sum_{j=0}^{n-1} 0,a_{n-1-j}\ldots a_0 s_j} |s\rangle$$

$$= \frac{1}{\sqrt{q}} \sum_{s=0}^{q} \underbrace{e^{2\pi i \sum_{j=0}^{n-1} a_{n-1}\ldots a_{n-1-j+1} s_j}}_{=1} e^{2\pi i \sum_{j=0}^{n-1} 0,a_{n-1-j}\ldots a_0 s_j} |s\rangle$$

$$= \frac{1}{\sqrt{q}} \sum_{s=0}^{q} (e^{2\pi i \sum_{j=0}^{n-1} a_{n-1}\ldots a_{n-1-j+1} s_j + 2\pi i \sum_{j=0}^{n-1} 0,a_{n-1-j}\ldots a_0 s_j}) |s\rangle$$

$$= \frac{1}{\sqrt{q}} \sum_{s=0}^{q} (e^{2\pi i \sum_{j=0}^{n-1} (a_{n-1}\ldots a_{n-1-j+1} + 0,a_{n-1-j}\ldots a_0) s_j}) |s\rangle$$

$$= \frac{1}{\sqrt{q}} \sum_{s=0}^{q} (e^{2\pi i \sum_{j=0}^{n-1} a_{n-1}\ldots a_{n-1-j+1},a_{n-1-j}\ldots a_0 s_j}) |s\rangle$$

$$= \frac{1}{\sqrt{q}} \sum_{s=0}^{q} (e^{2\pi i \sum_{j=0}^{n-1} (0,a_{n-1}\ldots a_0 2^j) s_j}) |s\rangle$$

$$= \frac{1}{\sqrt{q}} \sum_{s=0}^{q} (e^{2\pi i 0,a_{n-1}\ldots a_0 \sum_{j=0}^{n-1} 2^j s_j}) |s\rangle$$

$$= \frac{1}{\sqrt{q}} \sum_{s=0}^{q} (e^{2\pi i 0,a_{n-1}\ldots a_0 s}) |s\rangle$$

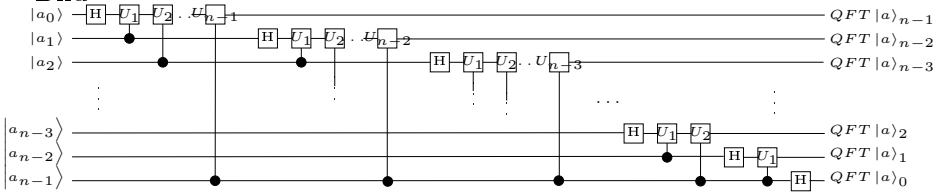
$$= \frac{1}{\sqrt{q}} \sum_{s=0}^{q} (e^{2\pi i \frac{a}{q} s}) |s\rangle$$

$$= \frac{1}{\sqrt{q}} \sum_{s=0}^{q} (e^{2\pi i a s/q}) |s\rangle = QFT_{\log_2 q} |a\rangle$$

Für ein Qubit $|a\rangle$ ist $U_{\text{Hadamard}} |a\rangle = |0\rangle + (-1)^a |1\rangle = |0\rangle + e^{\pi i a} |1\rangle = |0\rangle + e^{2\pi i 0,a} |1\rangle$. Sei $U_n(|c,t\rangle) = |0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes e^{2\pi i/2^{n+1}} \mathbb{1}$ ein kontrolliertes Phasengatter. Dann ist die Quantenfouriertransformation

$$QFT_{2^n} = \prod_{k=0}^{n-1} (\prod_{j=1}^{n-1-k} U_j^{(j+k,k)} U_{\text{Hadamard}}^{(k)})$$

wobei die Qubits vertauscht werden, also aus a_0 wird a_{n-1}, aus a_1 wird a_{n-2} usw. Als Bild

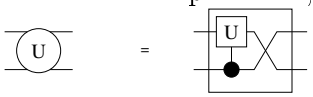


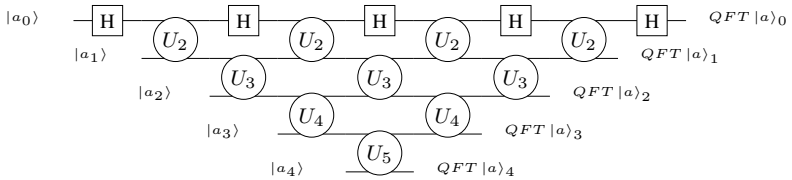
Quantenfouriertransformation[4]

Es gilt also

$$\begin{aligned} & QFT_{2^n} |a\rangle \\ = & \prod_{k=0}^{n-1} (\prod_{j=1}^{n-1-k} U_j^{(j+k,k)} U_{\text{Hadamard}}^{(k)}) |a\rangle \\ = & \frac{1}{\sqrt{q}} \bigotimes_{k=0}^{n-1} (|0\rangle + \prod_{j=0}^{k} e^{2\pi i/2^{j+1} a_{k-j}} |1\rangle) \end{aligned}$$

Dieses Quantennetzwerk ist äquivalent mit folgendem, das vor allem aus kontrollierten Phasengattern mit Swap besteht, die sehr kleine Messmuster besitzen.[5]





Quantenfouriertransformation[6]

2.3.4.1 Optimierung der QFT nach Cleve/Waltrous

Wie Cleve/Waltrous in [7] zeigen, lassen sich Netzwerke zur Quantenfouriertransformation, wenn ein Fehler zugelassen wird (der sich in der Praxis vermutlich sowieso nicht

[4] Korrigiert aus [9], S.118
[5] von Dan Browne
[6] aus [5]

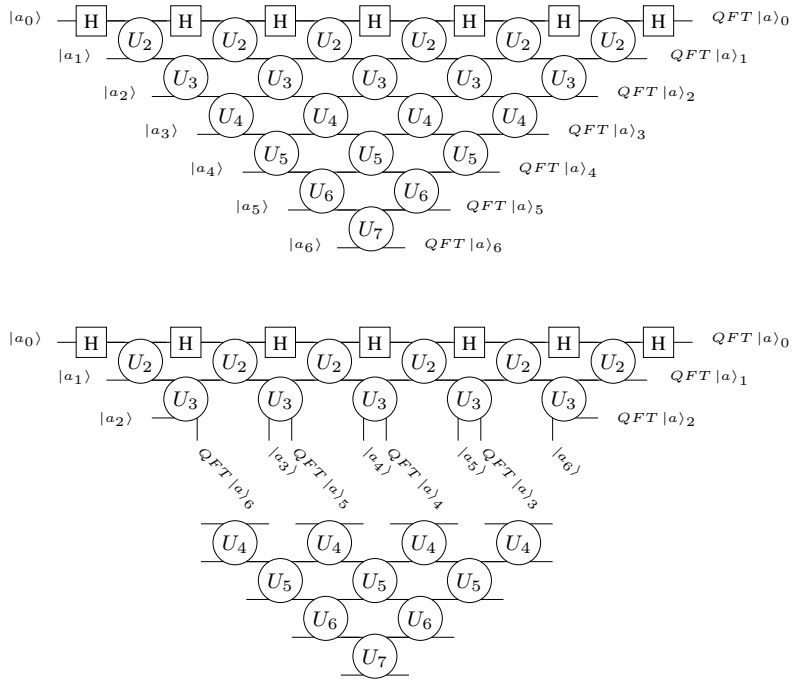


Abbildung 2.4: Verkleinerung der Quantenfouriertransformation nach Cleve/Waltrous

vermeiden lässt), massiv verkleinern. Betrachtet man obiges Quantennetzwerk, sind dort sehr viele Gatter zu erkennen, die nur kleine Phasendrehungen ausführen. Durch Weglassen von diesen erhält man ein $\mathcal{O}(n \log n)$ großes Netzwerk anstelle des exakten $\mathcal{O}(n^2)$ großen.

Genauer an Hand des hier verwendeten dreieckigen Netzwerkes aus kontrollierten Phasengattern mit Swap.

Behält man bei diesem Netzwerk nur die oberen k Reihen bei und schneidet den Zipfel aus den unteren $n-k$ Reihen ab (siehe Bild 2.4), die entstandene Transformation nenne ich $errQFT_{2^n}^k$, so ist der Fehler des Ausgangsqubits m mit dem Zustand $a^{(m)}$ gleich (zur

einfacheren Notation sei $U_0^{(x,x)} := U^{(x)}_{\text{Hadamard}}$, damit ist $QFT = \prod_{k=0}^{n-1} \prod_{j=0}^{n-1-k} U_j^{(j+k,k)}$):

$$
\begin{aligned}
& \left\| QFT_{2^n} \left|a^{(m)}\right\rangle - errQFT^k_{2^n} \left|a^{(m)}\right\rangle \right\| \\
= & \left\| \prod_{l=0}^{n-1} \prod_{j=0}^{n-1-l} U_j^{(j+l,l)} \left|a^{(m)}\right\rangle - \prod_{l=0}^{n-1} \prod_{j=0}^{\min(k-1,n-1-l)} U_j^{(j+l,l)} \left|a^{(m)}\right\rangle \right\| \\
= & \left\| \prod_{j=0}^{n-1-m} U_j^{(j+m,m)} \left|a^{(m)}\right\rangle - \prod_{j=0}^{\min(k-1,n-1-m)} U_j^{(j+m,m)} \left|a^{(m)}\right\rangle \right\| \\
= & \left\| \prod_{j=0}^{\min(k-1,n-1-m)} U_j^{(j+m,m)} \Big(\prod_{j=\min(k-1,n-1-m)+1}^{n-1-m} U_j^{(j+m,m)} \left|a^{(m)}\right\rangle - \left|a^{(m)}\right\rangle \Big) \right\| \\
\leq & \left\| \prod_{j=0}^{\min(k-1,n-1-m)} U_j^{(j+m,m)} \right\| \left\| \prod_{j=k}^{n-1-m} U_j^{(j+m,m)} \left|a^{(m)}\right\rangle - \left|a^{(m)}\right\rangle \right\| \\
= & \left\| \prod_{j=k}^{n-1-m} U_j^{(j+m,m)} \left|a^{(m)}\right\rangle - \left|a^{(m)}\right\rangle \right\| \\
\leq & \left\| \prod_{j=k}^{n-1-m} e^{2\pi i/2^{j+1}} \left|a^{(m)}\right\rangle - \left|a^{(m)}\right\rangle \right\| \\
= & \left\| (e^{2\pi i \sum_{j=k}^{n-1-m} 1/2^{j+1}} - 1) \left|a^{(m)}\right\rangle \right\|
\end{aligned}
$$

$$
\begin{aligned}
& \leq \left| (e^{2\pi i \sum_{j=k}^{n-1-m} 1/2^{j+1}} - 1) \right| \left\| \left|a^{(m)}\right\rangle \right\| \\
& \leq \left| (e^{2\pi i \sum_{j=k}^{n-1-m} 1/2^{j+1}} - 1) \right| \\
& \leq 2\pi \sum_{j=k}^{n-1-m} 1/2^{j+1} \\
& = 2\pi \Big(\sum_{j=0}^{n-1-m} 1/2^{j+1} - \sum_{j=0}^{k-1} 1/2^{j+1} \Big) \\
& = 2\pi (1 - 2^{-(n-1-m+1)} - 1 + 2^{-(k-1+1)}) \\
& = 2\pi (-2^{-(n-1-m+1)} + 2^{-k}) \\
& \leq \pi (2^{-k-1})
\end{aligned}
$$

Der Fehler nimmt also exponentiell ab. Der maximale Fehler eines Qubits (des obersten) ist kleiner als 2^{-k}, die maximale Distanz der unitären Transformationen nach der

Matrixnorm wird analog berechnet:

$$\left\| \prod_{l=0}^{n-1} \prod_{j=0}^{n-1-l} U_j^{(j+l,l)} - \prod_{l=0}^{n-1} \prod_{j=0}^{\min(k-1,n-1-l)} U_j^{(j+l,l)} \right\| \leq \pi(2^{-k-1})$$

Die Tiefe dieses Netzwerkes ist auch nur noch $\log n$.

2.3.5 Vollständigkeit bestimmter Gattermengen

Gattermengen, mit denen sich durch Zusammenschalten jede beliebige unitäre Transformation darstellen lässt, nennt man universelle Sätze. Der praktisch bedeutendste ist {CNOT, 1-Qubit-Rotation} (mit ihm wurde auch die Äquivalenz der Berechenbarkeit von Einweg-Quantencomputern und "üblichem" Quantencomputer bewiesen).

Fast alle Gattermengen sind vollständig: Aus einem Phasengatter mit Drehwinkel ϕ lassen sich - falls ϕ/π irrational ist - durch Hintereinanderschaltung Phasengatter mit jedem beliebigen Drehwinkel beliebig genau approximieren. Da nur für abzählbar viele ϕ ϕ/π rational ist, erfüllen "fast alle" Gatter diese Bedingung und machen damit fast alle Gattermengen vollständig. Das ist natürlich nur von theoretischem Interesse, da die Netzwerke unsinnig groß wären und die implizit angenomme beliebige Genauigkeit in der Wirklichkeit nicht vorkommt.

3 Das Modell des Einweg-Quantencomputers

3.1 Clusterzustand, Ising-Wechselwirkung, physikalische Realisierung

3.1.1 Clusterzustand[1]

Ein (zweidimensionaler) Clusterzustand $|\phi_C\rangle$ ist abstrakt definiert als ein quantenmechanischer Zustand, der die Eigenwertgleichungen

$$K^{(x,y)} |\phi_C\rangle = \kappa^{(x,y)} |\phi_C\rangle$$

mit den Korrelationsoperatoren

$$K^{(x,y)} = \sigma_{\mathrm{x}}^{(x,y)} \prod_{j \in \{(x+1,y),(x-1,y),(x,y+1),(x,y-1)\}} \sigma_{\mathrm{z}}^{(j)}$$

für alle $(x, y) \in \mathbb{Z}^2$ erfüllt, womit er auch vollständig charakterisiert ist.

Das Vorzeichen $\kappa^{(x,y)}$ hängt davon ab, welche Nachbarplätze von (x, y) besetzt sind: Mit $F^{(x,y)} = \begin{cases} 1, & \text{falls } (x,y) \text{ leer} \\ -1, & \text{falls } (x,y) \text{ besetzt} \end{cases}$ gilt $\kappa^{(x,y)} = F^{(x-1,y)} F^{(x,y-1)}$.

Dieser Zustand lässt sich als eine Ressource für die Berechnungen eines Quantencomputers betrachten. Er ist hochgradig verschränkt und durch Messungen reduziert man diese Verschränkung und prägt dem Zustand das Berechnungsproblem auf. Ist die Berechnung erfolgt, muss der Zustand neu erzeugt werden, deshalb heisst dieses Modell *Einweg*-Quantencomputer (siehe [11]).

Realisieren lässt sich ein Clusterzustand durch eine isingartige Wechselwirkung, die einen separierten Vielteilchenquantenzustand $\prod_a |+\rangle^{(a)}$ global verschränkt.

Die beschreibende Hamiltonsche Gleichung

$$H_{\mathrm{int}} = \hbar g(t) \sum_{a,a' \text{ Nachbarn}} \frac{\mathbb{1}^{(a)} + \sigma_{\mathrm{z}}^{(a)}}{2} \frac{\mathbb{1}^{(a')} - \sigma_{\mathrm{z}}^{(a')}}{2}$$

mit $g(t)$ einer extern kontrollierbaren, d.h. an- und abstellbaren, Komponente führt bei entsprechender Wahl von $g(t)$ zu einer Transformation

$$U = e^{i\varphi \sum_{a,a' \text{ Nachbarn}} \frac{\mathbb{1}^{(a)} + \sigma_{\mathrm{z}}^{(a)}}{2} \frac{\mathbb{1}^{(a')} - \sigma_{\mathrm{z}}^{(a')}}{2}}$$

[1] nach [4]

mit $\varphi = \int dt g(t)$ die für $\varphi = \pi$ aus dem Zustand mit allen Teilchen in $|+\rangle$ einen Clusterzustand $|\phi_C\rangle$ erzeugt, also $|\phi_C\rangle = U \prod_a |+\rangle^{(a)}$.

(H_{int} erzeugt bis auf lokale Drehungen aus einem gegebenen Anfangszustand die gleichen Zustände wie $H'_{\text{int}} = \hbar g(t) \sum_{a,a' \text{ Nachbarn}} \sigma_{\text{z}}^{(a)} \sigma_{\text{z}}^{(a')}$, eine Hamiltonsche Gleichung in der Ising-Form.)

Es folgt der Beweis für den Fall eines eindimensionalen Clusters, dass $U \prod_a |+\rangle^{(a)}$ ein Clusterzustand ist, d.h. $K^{(a)} U \prod_b |+\rangle^{(b)} = \kappa^{(a)} U \prod_b |+\rangle^{(b)}$ für alle a.

$$
\begin{aligned}
& K^{(a)} U \prod_b |+\rangle^{(b)} = \sigma_{\text{z}}^{(a-1)} \sigma_{\text{x}}^{(a)} \sigma_{\text{z}}^{(a+1)} U \prod_b |+\rangle^{(b)} \\
= \;& \sigma_{\text{z}}^{(a-1)} \sigma_{\text{x}}^{(a)} \sigma_{\text{z}}^{(a+1)} e^{i\pi \sum_{c,c' \text{ rechter Nachbar von } c} \frac{1^{(c)}+\sigma_{\text{z}}^{(c)}}{2} \frac{1^{(c')}-\sigma_{\text{z}}^{(c')}}{2}} \prod_b |+\rangle^{(b)} \\
= \;& \sigma_{\text{z}}^{(a-1)} \sigma_{\text{x}}^{(a)} \sigma_{\text{z}}^{(a+1)} e^{i\pi \sum_c \frac{1^{(c)}+\sigma_{\text{z}}^{(c)}}{2} \frac{1^{(c+1)}-\sigma_{\text{z}}^{(c+1)}}{2}} \prod_b |+\rangle^{(b)} \\
= \;& \sigma_{\text{x}}^{(a)} e^{i\pi \sum_c \frac{1^{(c)}+\sigma_{\text{z}}^{(c)}}{2} \frac{1^{(c+1)}-\sigma_{\text{z}}^{(c+1)}}{2}} \sigma_{\text{z}}^{(a-1)} \sigma_{\text{z}}^{(a+1)} \prod_b |+\rangle^{(b)} \\
= \;& e^{i\pi \sum_{c \neq a, a-1} \frac{1^{(c)}+\sigma_{\text{z}}^{(c)}}{2} \frac{1^{(c+1)}-\sigma_{\text{z}}^{(c+1)}}{2}} \sigma_{\text{x}}^{(a)} e^{i\pi \sum_{c=a,a-1} \frac{1^{(c)}+\sigma_{\text{z}}^{(c)}}{2} \frac{1^{(c+1)}-\sigma_{\text{z}}^{(c+1)}}{2}} \sigma_{\text{z}}^{(a-1)} \sigma_{\text{z}}^{(a+1)} \prod_b |+\rangle^{(b)} \\
= \;& e^{i\pi \sum_{c \neq a, a-1} \frac{1^{(c)}+\sigma_{\text{z}}^{(c)}}{2} \frac{1^{(c+1)}-\sigma_{\text{z}}^{(c+1)}}{2}} \\
& \sigma_{\text{x}}^{(a)} e^{i\pi (\frac{1^{(a)}+\sigma_{\text{z}}^{(a)}}{2} \frac{1^{(a+1)}-\sigma_{\text{z}}^{(a+1)}}{2} + \frac{1^{(a-1)}+\sigma_{\text{z}}^{(a-1)}}{2} \frac{1^{(a)}-\sigma_{\text{z}}^{(a)}}{2})} \sigma_{\text{z}}^{(a-1)} \sigma_{\text{z}}^{(a+1)} \prod_b |+\rangle^{(b)}
\end{aligned}
$$

mit $\sigma_{\text{x}} \sigma_{\text{z}} = -\sigma_{\text{z}} \sigma_{\text{x}}$

$$
\begin{aligned}
= \;& e^{i\pi \sum_{c \neq a, a-1} \frac{1^{(c)}+\sigma_{\text{z}}^{(c)}}{2} \frac{1^{(c+1)}-\sigma_{\text{z}}^{(c+1)}}{2}} \\
& e^{i\pi (\frac{1^{(a)}-\sigma_{\text{z}}^{(a)}}{2} \frac{1^{(a+1)}-\sigma_{\text{z}}^{(a+1)}}{2} + \frac{1^{(a-1)}+\sigma_{\text{z}}^{(a-1)}}{2} \frac{1^{(a)}+\sigma_{\text{z}}^{(a)}}{2})} \sigma_{\text{x}}^{(a)} \sigma_{\text{z}}^{(a-1)} \sigma_{\text{z}}^{(a+1)} \prod_b |+\rangle^{(b)}
\end{aligned}
$$

wegen $\sigma_{\text{x}} |+\rangle = |-\rangle$ und $\sigma_{\text{z}}^{(a-1)} \sigma_{\text{z}}^{(a+1)} = e^{-i\pi/2} e^{i\pi/2 \sigma_{\text{z}}^{(a-1)} \sigma_{\text{z}}^{(a+1)}}$ (Entwicklung)

$$
\begin{aligned}
= \;& e^{i\pi \sum_{c \neq a, a-1} \frac{1^{(c)}+\sigma_{\text{z}}^{(c)}}{2} \frac{1^{(c+1)}-\sigma_{\text{z}}^{(c+1)}}{2}} \\
& e^{i\pi (\frac{1^{(a)}-\sigma_{\text{z}}^{(a)}}{2} \frac{1^{(a+1)}-\sigma_{\text{z}}^{(a+1)}}{2} + \frac{1^{(a-1)}+\sigma_{\text{z}}^{(a-1)}}{2} \frac{1^{(a)}+\sigma_{\text{z}}^{(a)}}{2})} \\
& e^{-i\pi/2} e^{i\pi/2 \sigma_{\text{z}}^{(a-1)} \sigma_{\text{z}}^{(a+1)}} \prod_b |+\rangle^{(b)}
\end{aligned}
$$

$$
\begin{aligned}
&= e^{i\pi \sum_c \frac{\mathbb{1}^{(c)}+\sigma_{\mathrm{z}}^{(c)}}{2} \frac{\mathbb{1}^{(c+1)}-\sigma_{\mathrm{z}}^{(c+1)}}{2}} \\
&e^{i\pi - \frac{\mathbb{1}^{(a)}+\sigma_{\mathrm{z}}^{(a)}}{2} \frac{\mathbb{1}^{(a+1)}-\sigma_{\mathrm{z}}^{(a+1)}}{2}} e^{i\pi - \frac{\mathbb{1}^{(a-1)}+\sigma_{\mathrm{z}}^{(a-1)}}{2} \frac{\mathbb{1}^{(a)}-\sigma_{\mathrm{z}}^{(a)}}{2}} \\
&e^{i\pi(\frac{\mathbb{1}^{(a)}-\sigma_{\mathrm{z}}^{(a)}}{2} \frac{\mathbb{1}^{(a+1)}-\sigma_{\mathrm{z}}^{(a+1)}}{2} + \frac{\mathbb{1}^{(a-1)}+\sigma_{\mathrm{z}}^{(a-1)}}{2} \frac{\mathbb{1}^{(a)}+\sigma_{\mathrm{z}}^{(a)}}{2})} \\
&e^{-i\pi/2} \; e^{i\pi(\frac{\sigma_{\mathrm{z}}^{(a-1)} \sigma_{\mathrm{z}}^{(a+1)}}{2})} \prod_b |+\rangle^{(b)}
\end{aligned}
$$

$$
\begin{aligned}
&= e^{i\pi \sum_c \frac{\mathbb{1}^{(c)}+\sigma_{\mathrm{z}}^{(c)}}{2} \frac{\mathbb{1}^{(c+1)}-\sigma_{\mathrm{z}}^{(c+1)}}{2}} \\
&e^{i\pi/4(-\mathbb{1}^{(a,a+1)}+\sigma_{\mathrm{z}}^{(a+1)}-\sigma_{\mathrm{z}}^{(a)}+\sigma_{\mathrm{z}}^{(a)}\sigma_{\mathrm{z}}^{(a+1)}-\mathbb{1}^{(a-1,a)}+\sigma_{\mathrm{z}}^{(a)}-\sigma_{\mathrm{z}}^{(a-1)}+\sigma_{\mathrm{z}}^{(a-1)}\sigma_{\mathrm{z}}^{(a)})} \\
&e^{i\pi/4(\mathbb{1}^{(a,a+1)}-\sigma_{\mathrm{z}}^{(a+1)}-\sigma_{\mathrm{z}}^{(a)}+\sigma_{\mathrm{z}}^{(a)}\sigma_{\mathrm{z}}^{(a+1)}+\mathbb{1}^{(a-1,a)}+\sigma_{\mathrm{z}}^{(a)}-\sigma_{\mathrm{z}}^{(a-1)}+\sigma_{\mathrm{z}}^{(a-1)}\sigma_{\mathrm{z}}^{(a)})} \\
&e^{-i\pi/2} \; e^{i\pi/2(\sigma_{\mathrm{z}}^{(a-1)}\sigma_{\mathrm{z}}^{(a+1)})} \prod_b |+\rangle^{(b)}
\end{aligned}
$$

$$
\begin{aligned}
&= e^{i\pi \sum_c \frac{\mathbb{1}^{(c)}+\sigma_{\mathrm{z}}^{(c)}}{2} \frac{\mathbb{1}^{(c+1)}-\sigma_{\mathrm{z}}^{(c+1)}}{2}} \\
&e^{i\pi/2(\frac{\mathbb{1}^{(a-1)}+\sigma_{\mathrm{z}}^{(a-1)}}{2} \frac{\mathbb{1}^{(a)}+\sigma_{\mathrm{z}}^{(a)}}{2} + \frac{\mathbb{1}^{(a)}-\sigma_{\mathrm{z}}^{(a)}}{2} \frac{\mathbb{1}^{(a+1)}-\sigma_{\mathrm{z}}^{(a+1)}}{2} - \frac{\mathbb{1}^{(a)}-\sigma_{\mathrm{z}}^{(a)}}{2} \frac{\mathbb{1}^{(a+1)}-\sigma_{\mathrm{z}}^{(a+1)}}{2})} \\
&e^{i\pi(-\frac{\mathbb{1}^{(a)}+\sigma_{\mathrm{z}}^{(a)}}{2} \frac{\mathbb{1}^{(a+1)}-\sigma_{\mathrm{z}}^{(a+1)}}{2} + \frac{\mathbb{1}^{(a,a+1)}-\sigma_{\mathrm{z}}^{(a)}\sigma_{\mathrm{z}}^{(a+1)}}{2})} \prod_b |+\rangle^{(b)}
\end{aligned}
$$

Sei

$$
\begin{aligned}
R \;=\; & \frac{\mathbb{1}^{(a-1)}+\sigma_{\mathrm{z}}^{(a-1)}}{2} \frac{\mathbb{1}^{(a)}+\sigma_{\mathrm{z}}^{(a)}}{2} + \frac{\mathbb{1}^{(a)}-\sigma_{\mathrm{z}}^{(a)}}{2} \frac{\mathbb{1}^{(a+1)}-\sigma_{\mathrm{z}}^{(a+1)}}{2} - \frac{\mathbb{1}^{(a)}-\sigma_{\mathrm{z}}^{(a)}}{2} \frac{\mathbb{1}^{(a+1)}-\sigma_{\mathrm{z}}^{(a+1)}}{2} \\
& - \frac{\mathbb{1}^{(a)}+\sigma_{\mathrm{z}}^{(a)}}{2} \frac{\mathbb{1}^{(a+1)}-\sigma_{\mathrm{z}}^{(a+1)}}{2} + \frac{\mathbb{1}^{(a,a+1)}-\sigma_{\mathrm{z}}^{(a)}\sigma_{\mathrm{z}}^{(a+1)}}{2}
\end{aligned}
$$

Damit ist

$$
= \quad e^{i\pi \sum_c \frac{\mathbb{1}^{(c)}+\sigma_{\mathrm{z}}^{(c)}}{2} \frac{\mathbb{1}^{(c+1)}-\sigma_{\mathrm{z}}^{(c+1)}}{2}} e^{i\pi R} \prod_b |+\rangle^{(b)}
$$

Durch Nachrechnen für alle $i,j,k \in \{0,1\}$ erhält man $R\left|i^{(a-1)}j^{(a)}k^{(a+1)}\right\rangle =$

$\left|i^{(a-1)}j^{(a)}k^{(a+1)}\right\rangle$. Damit ist auch

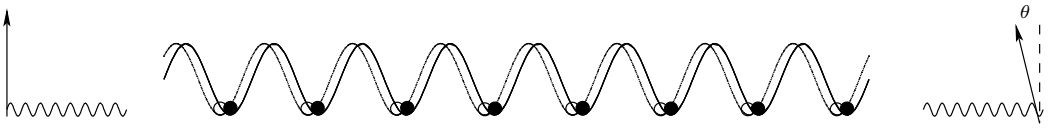
$$\begin{aligned}
& e^{i\pi R}\left|i^{(a-1)}j^{(a)}k^{(a+1)}\right\rangle \\
= & \sum_{n=0}^{\infty}(i\pi R)^n/n!\left|i^{(a-1)}j^{(a)}k^{(a+1)}\right\rangle \\
= & \sum_{n=0}^{\infty}\frac{(i\pi R)^n\left|i^{(a-1)}j^{(a)}k^{(a+1)}\right\rangle}{n!} \\
= & \sum_{n=0}^{\infty}\frac{(i\pi)^n R^n\left|i^{(a-1)}j^{(a)}k^{(a+1)}\right\rangle}{n!} \\
= & \sum_{n=0}^{\infty}\frac{(i\pi)^n}{n!}\left|i^{(a-1)}j^{(a)}k^{(a+1)}\right\rangle \\
= & e^{i\pi}\left|i^{(a-1)}j^{(a)}k^{(a+1)}\right\rangle
\end{aligned}$$

Schliesslich

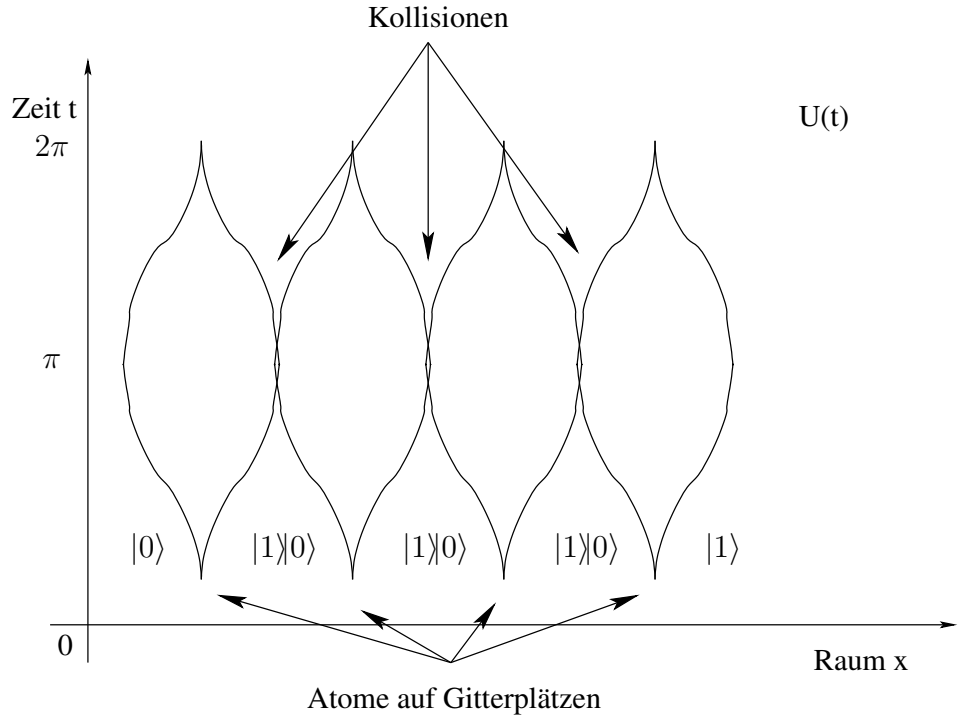
$$= -e^{i\pi\sum_c \frac{1^{(c)}+\sigma_z^{(c)}}{2}\frac{1^{(c+1)}-\sigma_z^{(c+1)}}{2}}\prod_b|+\rangle^{(b)}$$

Für zwei- und höherdimensionale Cluster verläuft die Rechnung analog, nur die zu betrachtenden Terme werden mehr.

3.1.2 Physikalische Realisierung



Ein Implementierungsvorschlag verwendet ultrakalte neutrale Atome, die in einem optischen Gitter - also periodischen Mikropotenzialen - aus zwei linear polarisierten Lasern gefangen sind. Sie werden in eine Superposition zweier Feinstrukturzustände versetzt und es wird durch Winkelverschiebung an den Lasern ein zustandsabhängiges Potential erzeugt. Dabei bewegen sich Atome in ihrer Potenzialwelle im Zustand $|0\rangle$ nach links, solche im Zustand $|1\rangle$ nach rechts und es kommt zu kontrollierten Kollisionen (nach [10]).



Diese Systeme verbinden gute Kohärenzeigenschaften mit Skalierbarkeit. Denn ist die relative Häufigkeit der besetzten Potentialsenken größer als die durch die Perkolationstheorie bestimmte Schranke, dann ist mit großer Wahrscheinlichkeit der Cluster ähnlich groß wie die Atomfalle. Es wurden schon für dreidimensionale Gitter Füllraten von 44% erreicht, was deutlich oberhalb der Perkolationsschranke von 31% ist. Ein Problem ist allerdings nach wie vor - wie in anderen Realisierungen von Quantencomputern - der Zugriff auf die einzelnen Qubits, hier also Atome.

Eine andere Möglichkeit, eine Ising-Wechselwirkung zu realisieren, beruht auf Arrays von kapazitiv gekoppelten quantum dots und könnte auch als Grundlage für eine Implementierung des Einwegquantencomputers dienen (nach [4, 11]).

Experimentell sind Grundlagen z.B. in den Bose-Einstein-Kondensationsexperimenten gelegt worden, wo es gelang eine relativ große Zahl von ultrakalten Atomen herzustellen. Eine weitergehende experimentelle Erforschung dieser Zusammenhänge ist - mit und ohne Bezug zu Quantencomputern - derzeit im Gange. Z.B. wurden in der Hänsch-Gruppe neue Experimente zu "Mott-Phasen" von Bose-Einstein-Kondensaten durchgeführt[2].

Diese Arbeit setzt vorraus, dass man beliebig große und regelmässige (ohne Löcher und andere Fehlstellen, wobei diese Notwendigkeit ohne grössere Schwierigkeiten aufzuheben ist) zweidimensionale Cluster erzeugen und manipulieren kann.

[2] siehe [8] und www.mpq.mpg.de/~haensch/bec/

4 Quantengatter und Messvorgänge

In diesem Abschnitt wird der Zusammenhang zwischen Quantengattern und den sie implementierenden Messmustern dargestellt, und für einen Satz von Elementargattern die entsprechenden Messmuster entworfen. Aus diesen werden dann die im Abschnitt 2.3.3 vorgestellten Quantennetzwerke als Messmuster zusammengebaut.

Es soll zunächst bewiesen werden, dass das Messmuster $\mathcal{M}$ eine unitäre Transformation $U' = UU_\Sigma$ zwischen den Eingabe- und den Ausgabequbits auf dem Cluster realisiert. Also: $|\Phi_{\text{out}}\rangle = U' |\Phi_{\text{in}}\rangle$. Dabei ist U_Σ ein "Beiprodukt" aus σ_{x}- und σ_{z}-Rotationen, das sich aus den zufälligen Messergebnissen ergibt. Es muss durch Anpassen der schiefen Messrichtungen und bei der Ergebnisbewertung berücksichtigt werden.

Das geschieht über einen vermeintlichen Umweg, nämlich zuerst zu zeigen, dass der gemessene Clusterzustand bestimmte Eigenwertgleichungen erfüllt und dann aus diesen auf die Durchführung von U' zu schliessen. Dem ersten Teil - für einige elementare Gatter zu belegen, dass ein Quantenzustand nach dem Messen mit dem zugehörigen Muster bestimmte Korrelationsrelationen erfüllt - werde ich mich im übernächten Punkt widmen. Für den zweiten Teil gibt es einen allgemeinen Beweis (ohne den das ganze Verfahren auch leicht obskur wäre), der im Folgenden dargestellt wird.

4.1 Beweis des Zusammenhangs zwischen Messmustern und Quantengattern[1]

Sei "in" eine Menge von Qubits (eines Clusterzustandes) - die Eingangspins, out eine injektive Abbildung von in auf die Qubits des Clusterzustandes - die Zuordnung von Eingangs- zu Ausgangspins. Damit ist out(in), das Bild der Funktion "out", die Menge der Ausgangsqubits des Gatters. Falls dann gilt

$$\begin{aligned} \forall i \in \text{in}. \sigma_{\text{x}}^{(i)} (U^{(\text{out(in)})} \sigma_{\text{x}}^{(\text{out}(i))} U^{\dagger(\text{out(in)})}) \, |\Phi\rangle &= \pm |\Phi\rangle \\ \text{und } \forall i \in \text{in}. \sigma_{\text{z}}^{(i)} (U^{(\text{out(in)})} \sigma_{\text{z}}^{(\text{out}(i))} U^{\dagger(\text{out(in)})}) \, |\Phi\rangle &= \pm |\Phi\rangle \end{aligned}$$

kann man durch Messen in der σ_x-Basis der Qubits aus der Menge der Inputpins in die Transformation U durchführen, die Quanteninformation durch das vorher durch ein Messmuster hergestellte Gatter schieben.

Sei also

[1] stammt von Robert Raussendorf

1. $\forall i \in \text{in}.\sigma_{\text{x}}^{(i)}(U^{(\text{out(in)})}\sigma_{\text{x}}^{(\text{out}(i))}U^{\dagger(\text{out(in)})})\,|\Phi\rangle = \tau_i\,|\Phi\rangle$
2. $\forall i \in \text{in}.\sigma_{\text{z}}^{(i)}(U^{(\text{out(in)})}\sigma_{\text{z}}^{(\text{out}(i))}U^{\dagger(\text{out(in)})})\,|\Phi\rangle = \upsilon_i\,|\Phi\rangle$

Sei $|\phi_{\text{X}}\rangle$ der Zustand des auf die Qubitmenge X eingeschränkten Quantensystems. Misst man nun die Eingabequbits in in der x-Basis und erhält die Ergebnisse x_i, dann gilt $\left|\phi_{\text{out(in)}}\right\rangle = U'\,|\phi_{\text{in}}\rangle$ mit $U' = U^{(\text{in})}U_\Sigma$ und $U_\Sigma = \prod_{i\in\text{in}}(\sigma_z^{(i)})^{x_i}$.

Wegen der Linearität der Operatoren reicht es, die Aussage für $|\phi_{\text{in}}\rangle \in \{|z\rangle\,|z \text{ Vektor der Rechenbasis}\}$ zu zeigen, also

$$\begin{aligned}
|\phi_{\text{in}}\rangle &= \left|\sum_j \alpha_j z_j^{(in)}\right\rangle \\
\Rightarrow U'\,|\phi_{\text{in}}\rangle &= U'\left|\sum_j \alpha_j z_j^{(in)}\right\rangle \\
&=^{\text{Linearität}} \sum_j \alpha_j U'\left|z_j^{(in)}\right\rangle \\
&= \\
&=^{\text{noch zu zeigen!}} \sum_j \alpha_j \left|z_j^{(out(in))}\right\rangle \\
&=^{\text{Linearität}} \left|\phi_{\text{out(in)}}\right\rangle
\end{aligned}$$

.

Sei $|b\rangle := |\phi_{\text{in}}\rangle \in$ Rechenbasis, also darstellbar als $\bigotimes_{i=1}^n |z_i\rangle\,,\ z_i \in \{0,1\}$.

Sei $P_{\text{z}} = \bigotimes_{i\in\text{in}} \frac{1^{(i)}+(-1)^{z_i}\sigma_z^{(i)}}{2}$ der dazugehörige Projektor. Damit gilt $P_{\text{z}}\,|\phi\rangle = |\phi\rangle$, für alle $|\phi\rangle$ mit den in-Qubits $|\phi_{\text{in}}\rangle$ in der Rechenbasis, wie oben dargestellt, da der Projektor ja gerade in den Unterraum projiziert, in dem sich $|\phi\rangle$ bereits befindet.

Aus der 2.Vorraussetzung

$$\begin{aligned}
\forall i \in \text{in}.\sigma_{\text{z}}^{(i)}(U^{(\text{out(in)})}\sigma_{\text{z}}^{(\text{out}(i))}U^{\dagger(\text{out(in)})})\,|\Phi\rangle &= \upsilon_i\,|\Phi\rangle \\
\forall i \in \text{in}.\sigma_{\text{z}}^{(i)}(U^{(\text{out(in)})}\sigma_{\text{z}}^{(\text{out}(i))}U^{\dagger(\text{out(in)})})P_{\text{z}}\,|\Phi\rangle &= \upsilon_i P_{\text{z}}\,|\Phi\rangle \ \text{(einsetzen)} \\
\forall i \in \text{in}.(U^{(\text{out(in)})}\sigma_{\text{z}}^{(\text{out}(i))}U^{\dagger(\text{out(in)})})(-1)^{z_i}P_{\text{z}}\,|\Phi\rangle &= \upsilon_i P_{\text{z}}\,|\Phi\rangle \ \text{(wegen } \sigma_{\text{z}}^{(i)}P_{\text{z}} = (-1)^{z_i}P_{\text{z}}) \\
\forall i \in \text{in}.(U^{(\text{out(in)})}\sigma_{\text{z}}^{(\text{out}(i))}U^{\dagger(\text{out(in)})})P_{\text{z}}\,|\Phi\rangle &= (-1)^{z_i}\upsilon_i P_{\text{z}}\,|\Phi\rangle
\end{aligned}$$

Multiplikation mit $U^{\dagger(\text{out(in)})}$

$$\forall i \in \text{in}.(\sigma_{\text{z}}^{(\text{out}(i))}U^{\dagger(\text{out(in)})})P_{\text{z}}\,|\Phi\rangle = (-1)^{z_i}\upsilon_i P_{\text{z}}U^{\dagger(\text{out(in)})}\,|\Phi\rangle$$

Substitution $|\Phi'\rangle := U^{\dagger(\text{out(in)})}\,|\Phi\rangle$

$$\forall i \in \text{in}.\sigma_{\text{z}}^{(\text{out}(i))}P_{\text{z}}\left|\Phi'\right\rangle = (-1)^{z_i}\upsilon_i P_{\text{z}}\left|\Phi'\right\rangle$$

Hier sieht man nun, dass der Ausdruck $P_{\mathrm{z}} \left|\Phi'\right\rangle$ ein Eigenvektor von $\sigma_{\mathrm{z}}^{(\mathrm{out}(i))}$ für alle i ist, damit

$$P_{\mathrm{z}} \left|\Phi'\right\rangle_{\mathrm{out(in)}} = \bigotimes_{i=1}^{n} \left|\mathrm{o}_i\right\rangle_z e_z^{i\chi_{b,i}}$$

da normierte Eigenvektoren von σ_{z} die Form $e^{i\varphi} \left|r\right\rangle_z$, $r \in \{0,1\}$ haben.
Mit $\mathbb{1}^{(i)} = \prod_i (\sigma_z^{(i)})^{x_i} \prod_i (\sigma_z^{(i)})^{x_i}$ und $e^{i\chi_b} = \prod_{i=1}^{n} e^{i\chi_{b,i}}$ gilt dann

$$\left|\Phi'\right\rangle = e^{i\chi_b} \prod_i (\sigma_z^{(i)})^{x_i} \bigotimes_{i=1}^{n} (-1)^{x_i} \left|z\right\rangle_{z,i}$$

mit $\mathrm{e}^{i\chi_{b,x}} = e^{i\chi_b}(-1)^{x_i}$ und $\left|\phi_{\mathrm{in}}\right\rangle = \left|b\right\rangle = \bigotimes_{i=1}^{n} \left|z_i\right\rangle_{z,i}$

$$\left|\Phi'\right\rangle = e^{i\chi_{b,x}} \prod_i (\sigma_z^{(i)})^{x_i} \left|\phi_{\mathrm{in}}\right\rangle$$

mit der Definition von $\left|\Phi'\right\rangle$ erhält man das Ergebnis

$$\left|\Phi_{\mathrm{out}}\right\rangle = e^{i\chi_{b,x}} U \prod_i (\sigma_z^{(i)})^{x_i} \left|\phi_{\mathrm{in}}\right\rangle \quad (*)$$

Also, falls die Phasen $e^{i\chi_{b,x}}$ alle gleich sind, folgt die Behauptung.
Um das zu zeigen, betrachten wir zuerst den Spezialfall $\left|\Phi_{\mathrm{in}}\right\rangle = \otimes_{i=1}^{n} \left|+\right\rangle$: Es wird der Projektor $P = \frac{\mathbb{1}^{(i\in\mathrm{in})} + (-1)^{x_i} \sigma_x^{(i\in\mathrm{in})}}{2}$ für die $\sigma_x^{(i\in\mathrm{in})}$-Messungen angewandt, der - da $P\left|+\right\rangle = \left|+\right\rangle$ - den Zustand unverändert lässt. Damit gilt für den Zustand $\left|\Phi\right\rangle$

$$\left|\Phi\right\rangle = \bigotimes_{i\in\mathrm{in}} \frac{\mathbb{1}^{(i)} + (-1)^{x_i} \sigma_x^{(i)}}{2} \left|\Phi\right\rangle$$

,

Nach der 1.Korrelationsrelation $\forall i \in \mathrm{in}. \sigma_{\mathrm{x}}^{(i)} (U \sigma_{\mathrm{x}}^{(\mathrm{out}(i))} U^{\dagger}) \left|\Phi\right\rangle = \tau_i \left|\Phi\right\rangle$ folgt durch Substitution obiger Gleichung

$$\begin{aligned}
\tau_i \left|\Phi\right\rangle &= (\bigotimes_{j\in\mathrm{in}} \frac{\mathbb{1}^{(j)} + (-1)^{x_j} \sigma_x^{(j)}}{2}) \sigma_{\mathrm{x}}^{(i)} (U^{(\mathrm{out(in)})} \sigma_{\mathrm{x}}^{(\mathrm{out}(i))} U^{\dagger(\mathrm{out(in)})}) \left|\Phi\right\rangle \\
&= (-1)^{x_i} (U^{(\mathrm{out(in)})} \sigma_{\mathrm{x}}^{(\mathrm{out}(i))} U^{\dagger(\mathrm{out(in)})}) (\bigotimes_{j\in\mathrm{in}} \frac{\mathbb{1}^{(j)} + (-1)^{x_j} \sigma_x^{(j)}}{2}) \left|\Phi\right\rangle \\
&= (-1)^{x_i} (U \sigma_{\mathrm{x}}^{(\mathrm{out}(i))} U^{\dagger}) \left|\Phi\right\rangle
\end{aligned}$$

Substitution: $\left|\Phi'\right\rangle := U^{\dagger(\mathrm{out(in)})} \left|\Phi\right\rangle$ nach Multiplikation der Gleichung mit $U^{\dagger(\mathrm{out(in)})}$ ergibt

$$\left|\Phi'\right\rangle = (-1)^{x_i}\sigma_{\mathrm{x}}^{(out(i))}\left|\Phi\right\rangle$$

$$\text{somit } \left|\Phi'_{\text{out}}\right\rangle = \bigotimes_{i=1}^{n}\left|x_i\right\rangle_{x,i} e^{i\xi_i},\ \text{da } \left|\Phi'_{\text{out}}\right\rangle \text{ Eigenvektor von allen } \sigma_x^{(i)}$$

mit $\left|-\right\rangle = \sigma_z\left|+\right\rangle$ ist $\left|\Phi'_{\text{out}}\right\rangle = \bigotimes_{i=1}^{n}(\sigma_z^{(i)})^{x_i}\bigotimes_{j=1}^{n}\left|+\right\rangle e^{i\xi_i}$. Damit und mit $e^{i\xi} = \prod_{i=1}^{n} e^{i\xi_i}$

$$\left|\Phi_{\text{out}}\right\rangle = U\left|\Phi'_{\text{out}}\right\rangle = e^{i\xi}U\prod_{i=1}^{n}(\sigma_z^{(i)})^{x_i}\bigotimes_{j=1}^{n}\left|+\right\rangle$$

.

Mit $\left|\Phi_{\text{in}}\right\rangle = \bigotimes_{j=1}^{n}\left|+\right\rangle$ gilt

$$\left|\Phi_{\text{out}}\right\rangle = U\left|\Phi'_{\text{out}}\right\rangle = e^{i\xi}U\underbrace{\prod_{i=1}^{n}(\sigma_z^{(i)})^{x_i}}_{\text{Beiprodukt}}\left|\Phi_{\text{in}}\right\rangle$$

Also gilt die Behauptung, falls $\left|\Phi_{\text{in}}\right\rangle = \bigotimes_{j=1}^{n}\left|+\right\rangle$.
Damit lässt sich die noch offene Behauptung zeigen, alle $\chi_{b,x}$ seien gleich:
Sei $g = \left\langle\Phi_{\text{out}}\right| U'\bigotimes\left|+\right\rangle$ mit $U' = U\prod_i(\sigma_z^{(i)})^{x_i}$. Aus obiger Gleichnung ergibt sich

$$g = \left\langle\Phi_{\text{out}}\right| U'^{\dagger}e^{-i\xi}U'\bigotimes_{i=1}^{n}\left|+\right\rangle = e^{-i\xi}$$

woraus sowohl $|g| = 1$ folgt als auch

$$\begin{aligned} g &= \sum_b\sum_{b'}\frac{1}{2^n}\left\langle b\right| U'\left|b'\right\rangle \\ &= \sum_{b,b'}\left\langle b\right| U'^{\dagger}e^{-i\chi_{b,x}}U'\left|b'\right\rangle \\ &= {}^{b,b'\text{ orthonormal}}\frac{1}{2^n}\sum_b e^{-i\chi_{b,x}} \end{aligned}$$

mit $|g| = 1$ und $\frac{1}{2}\left|e^{i\varphi} + e^{i\delta}\right| = 1 \Rightarrow \varphi = \delta$ ergibt sich die Gleichheit aller $\chi_{b,x}$ und der Beweis ist komplett.

4.1.1 Verschränkungseigenschaft

Die Verschränkungsoperation kommutiert mit den Messoperationen auf anderen Qubits. Deshalb genügt für den Beweis des Funktionieren eines Gatters auch in einem Gesamtmessmuster seine isolierte Betrachtung. Ist nämlich seine Arbeitsweise isoliert nachgewiesen, dann funktioniert es auch in einem Zustand, bei dem die Atome, die nicht zu ihm gehören, noch unverschränkt sind. Man argumentiert folgendermassen: Führt man

die Gatter der Reihe nach aus, wobei immer nur die zum jeweils aktuellen gehörenden Qubits verschränkt werden, dann das Messmuster (mit Messung der Eingangspins in X) ausgeführt wird und dann das nächste Gatter an die Reihe kommt, dann entspricht die realisierte unitäre Transformation der der Zusammenschaltung der Einzelgatter. Weil sie kommutieren kann man die Verschränkungsoperationen alle an den Anfang ziehen.

4.2 Quantengatter des Einweg-Quantencomputers

Die die Messmuster darstellenden Bilder bestehen aus folgenden, auf einem kartesischen Gitter angeordneten, Elementen: Eingangsqubits  , Ausgangsqubits  , diese beiden Arten nenne ich in Analogie zu elektronischen Schaltungen (Ein- bzw. Ausgangs-)Pins. Ein Messmuster realisiert also eine unitäre Transformation zwischen dem Quantenzustand $|\Phi_{in}\rangle$ seiner Eingangspins und dem Quantenzustand $|\Phi_{out}\rangle$ seiner Ausgangspins.

Das eigentliche Muster besteht aus Messungen in der σ_x-Eigenbasis  , σ_y-Eigenbasis  und der σ_z-Eigenbasis  , sowie schiefen Messungen  , deren Messbasis die σ_x-Eigenbasis um einen Winkel φ um die z-Achse gedreht ist.

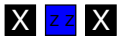
Leere Stellen sind auf dem Cluster entweder unbesetzt oder sie werden in σ_z herausgemessen. Es gibt also zu jedem Messmuster mit Leerstellen ein äquvalentes, bei dem manche dieser Stellen mit Qubits besetzt sind, wobei diese einfach mit σ_z -Messungen belegt werden. Was sich dabei ändert, sind die Beiprodukte U_Σ, in die die neuen Messwerte mit eingehen und die entsprechend propagiert werden müssen.

Die Darstellung der Korrelationsrelationen benutzt die Farbe Schwarz zur Markierung der Korrelationszentren( ,  , ), blau sind die σ_zs der Nachbarn( , ).

Wegen der Symmetrie der Eigenwertgleichungen lassen sich die Gatter beliebig spiegeln und in 90° Schritten drehen.

Die Form der Gatter, deren Ein- und Ausgänge immer einen geraden Abstand voneinander haben, induziert auf dem Clusterzustand eine Gitterstruktur, bei der ein Ausgang eines Gatters nur mit dem Eingang eines anderen verbunden werden kann, wenn der Abstand gerade ist, d.h. für (x_1, y_1) und (x_2, y_2): $x_1 - x_2 + y_1 - y_2 \equiv 0 \bmod 2$ ist. Ob das eine Grundlage in der physikalischen Struktur des Clusterzustandes hat, ist mir unbekannt.

4.2.1 Quantenkabel

 Messmuster |  X Korrelationen |  Z Korrelationen

Nochmal das oben abstrakt Eingeführte am konkreten Beispiel: Das linke Bild sagt, der Messprojektor des Quantenkabels besteht aus einer Messung des mittleren (2,1) Qubits in der σ_x-Eigenbasis. Zur Übertragung des Quantenzustand im Qubit des Eingangspins auf das Qubit des Ausgangspins muss noch das Eingangspin (1,1) in σ_x gemessen werden.

Zum Beweis dieser Tatsache werden die Korrelationsrelationen für X und Z benutzt. Nach obigen Bildern gilt mit Korrelationszentren in (1,1) und (3,1) die Eigenwertgleichung $\sigma_x^{(1,1)}\sigma_x^{(3,1)}|\phi\rangle = \kappa^{(1,1)}\kappa^{(3,1)}|\phi\rangle$, die für die X Gleichungen verwendet wird.

Für Z verwendet man die Gleichung mit dem Korrelationszentrum (2,1), sie lautet $\sigma_{\mathrm{z}}^{(1,1)}\sigma_{\mathrm{x}}^{(2,1)}\sigma_{\mathrm{z}}^{(3,1)}\left|\phi\right\rangle = \kappa^{(2,1)}\left|\phi\right\rangle$.

Diese Korrelationsrelationen erhält man, indem man die Eigenwertgleichungen der Korrelationszentren ineinander substituiert und ausmultipliziert. Wegen $\sigma_k\sigma_k = \mathbb{1}$ lassen sich doppelte σs streichen. Hier für die erste Gleichung:

$$\begin{aligned} & \sigma_{\mathrm{x}}^{(1,1)}\sigma_{\mathrm{z}}^{(2,1)}\left|\phi\right\rangle = \kappa^{(1,1)}\left|\phi\right\rangle,\ \sigma_{\mathrm{x}}^{(3,1)}\sigma_{\mathrm{z}}^{(2,1)}\left|\phi\right\rangle = \kappa^{(3,1)}\left|\phi\right\rangle \\ \Rightarrow\ & \sigma_{\mathrm{x}}^{(1,1)}\sigma_{\mathrm{z}}^{(2,1)}\sigma_{\mathrm{z}}^{(2,1)}\sigma_{\mathrm{x}}^{(3,1)}\left|\phi\right\rangle = \kappa^{(1,1)}\kappa^{(3,1)}\left|\phi\right\rangle \\ \Leftrightarrow\ & \sigma_{\mathrm{x}}^{(1,1)}\sigma_{\mathrm{x}}^{(3,1)}\left|\phi\right\rangle = \kappa^{(1,1)}\kappa^{(3,1)}\left|\phi\right\rangle \end{aligned}$$

. Die zweite Gleichung ist identisch mit einer Eigenwertgleichung des Clusterzustandes. Weiter mit der Berechnung der Korrelationsrelationen.

Sei $\mu^{(x,y)}$ der an der Stelle (x,y) gemessene Wert. Nun wendet man auf den Zustand den Messprojektor $\mathcal{P}_{\mathrm{Kabel}} = \frac{\mathbb{1}^{(2,1)}+\mu^{(2,1)}\sigma_{\mathrm{x}}^{(2,1)}}{2}$ an, der den Zustand in den nach der Messung verwandelt. Da Operationen auf unterschiedlichen Qubits miteinander kommutieren und mit $\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle = \left|\phi_{\mathrm{Kabel}}\right\rangle$, gilt:

$$\begin{aligned} & \mathcal{P}_{\mathrm{Kabel}}\sigma_{\mathrm{x}}^{(1,1)}\sigma_{\mathrm{x}}^{(3,1)}\left|\phi\right\rangle = \mathcal{P}_{\mathrm{Kabel}}\kappa^{(1,1)}\kappa^{(3,1)}\left|\phi\right\rangle \\ \Leftrightarrow\ & \sigma_{\mathrm{x}}^{(1,1)}\sigma_{\mathrm{x}}^{(3,1)}\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle = \kappa^{(1,1)}\kappa^{(3,1)}\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle \\ \Leftrightarrow\ & \sigma_{\mathrm{x}}^{(1,1)}\sigma_{\mathrm{x}}^{(3,1)}\left|\phi_{\mathrm{Kabel}}\right\rangle = \kappa^{(1,1)}\kappa^{(3,1)}\left|\phi_{\mathrm{Kabel}}\right\rangle \end{aligned}$$

und für

$$\begin{aligned} & \mathcal{P}_{\mathrm{Kabel}}\sigma_{\mathrm{z}}^{(1,1)}\sigma_{\mathrm{x}}^{(2,1)}\sigma_{\mathrm{z}}^{(3,1)}\left|\phi\right\rangle = \mathcal{P}_{\mathrm{Kabel}}\kappa^{(2,1)}\left|\phi\right\rangle \\ \Leftrightarrow\ & \sigma_{\mathrm{z}}^{(1,1)}\mathcal{P}_{\mathrm{Kabel}}\sigma_{\mathrm{x}}^{(2,1)}\sigma_{\mathrm{z}}^{(3,1)}\left|\phi\right\rangle = \kappa^{(2,1)}\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle \\ \Leftrightarrow\ & \sigma_{\mathrm{z}}^{(1,1)}\frac{\mathbb{1}^{(2,1)}+\mu^{(2,1)}\sigma_{\mathrm{x}}^{(2,1)}}{2}\sigma_{\mathrm{x}}^{(2,1)}\sigma_{\mathrm{z}}^{(3,1)}\left|\phi\right\rangle = \kappa^{(2,1)}\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle \\ \Leftrightarrow\ & \sigma_{\mathrm{z}}^{(1,1)}\frac{\sigma_{\mathrm{x}}^{(2,1)}+\mu^{(2,1)}\mathbb{1}^{(2,1)}}{2}\sigma_{\mathrm{z}}^{(3,1)}\left|\phi\right\rangle = \kappa^{(2,1)}\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle \\ \Leftrightarrow\ & \sigma_{\mathrm{z}}^{(1,1)}\mu^{(2,1)}\mathcal{P}_{\mathrm{Kabel}}\sigma_{\mathrm{z}}^{(3,1)}\left|\phi\right\rangle = \kappa^{(2,1)}\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle \\ \Leftrightarrow\ & \sigma_{\mathrm{z}}^{(1,1)}\sigma_{\mathrm{z}}^{(3,1)}\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle = \mu^{(2,1)}\kappa^{(2,1)}\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle \end{aligned}$$

.

Sei nun $U_{\mathrm{Kabel}} = \mathbb{1}$. Dann gilt nach den obigen Herleitungen
$\sigma_{\mathrm{x}}^{(1,1)}\sigma_{\mathrm{x}}^{(3,1)}\left|\phi_{\mathrm{Kabel}}\right\rangle = \kappa^{(1,1)}\kappa^{(3,1)}\left|\phi_{\mathrm{Kabel}}\right\rangle$, damit

$$\sigma_{\mathrm{x}}^{(1,1)}U_{\mathrm{Kabel}}^{(3,1)}\sigma_{\mathrm{x}}^{(3,1)}U_{\mathrm{Kabel}}^{\dagger(3,1)}\left|\phi_{\mathrm{Kabel}}\right\rangle = \kappa^{(1,1)}\kappa^{(3,1)}\left|\phi_{\mathrm{Kabel}}\right\rangle$$

und $\sigma_{\mathrm{z}}^{(1,1)}\sigma_{\mathrm{z}}^{(3,1)}\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle = \mu^{(2,1)}\kappa^{(2,1)}\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle$, damit

$$\begin{aligned} & \sigma_{\mathrm{z}}^{(1,1)}U_{\mathrm{Kabel}}^{(3,1)}\sigma_{\mathrm{z}}^{(3,1)}U_{\mathrm{Kabel}}^{\dagger(3,1)}\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle = \mu^{(2,1)}\kappa^{(2,1)}\mathcal{P}_{\mathrm{Kabel}}\left|\phi\right\rangle \\ \Leftrightarrow\ & \sigma_{\mathrm{z}}^{(1,1)}U_{\mathrm{Kabel}}^{(3,1)}\sigma_{\mathrm{z}}^{(3,1)}U_{\mathrm{Kabel}}^{\dagger(3,1)}\left|\phi_{\mathrm{Kabel}}\right\rangle = \mu^{(2,1)}\kappa^{(2,1)}\left|\phi_{\mathrm{Kabel}}\right\rangle \end{aligned}$$

.

Mit $\text{in} = \{(1,1)\}$ und $\text{out}((1,1)) = (3,1)$ erhalte ich dann das gewünschte Ergebnis

$$\forall i \in \text{in}. \sigma_{\text{x}}^{(i)} U_{\text{Kabel}}^{\text{out}(i)} \sigma_{\text{x}}^{\text{out}(i)} U_{\text{Kabel}}^{\dagger \text{out}(i)} \left|\phi_{\text{Kabel}}\right\rangle = \tau_{\text{x}}^{(i)} \left|\phi_{\text{Kabel}}\right\rangle$$

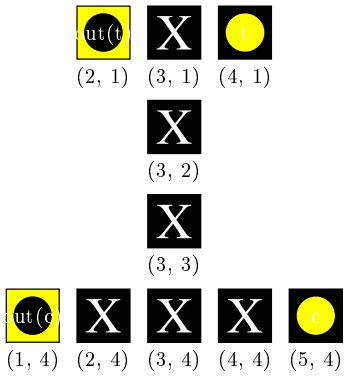
und

$$\forall i \in \text{in}. \sigma_{\text{z}}^{(i)} U_{\text{Kabel}}^{\text{out}(i)} \sigma_{\text{z}}^{\text{out}(i)} U_{\text{Kabel}}^{\dagger \text{out}(i)} \left|\phi_{\text{Kabel}}\right\rangle = \tau_{\text{z}}^{(i)} \left|\phi_{\text{Kabel}}\right\rangle$$

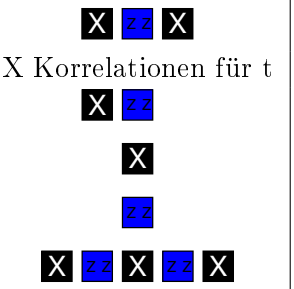
mit $\tau_{\text{x}}^{(i)} = \kappa^{(1,1)}\kappa^{(3,1)}$und $\tau_{\text{z}}^{(i)} = \mu^{(2,1)}\kappa^{(2,1)}$.

Aus dem Beweis 4.1 folgt nun, dass der Zustand des Qubits (1,1) nach seiner Messung in X zum Qubit (3,1) geleitet wird.

4.2.2 CNOT-Gatter

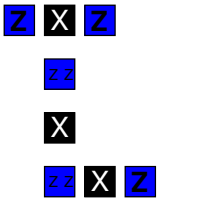


Messmuster



X Korrelationen für t

X Korrelationen für c



Z Korrelationen für t

Z Korrelationen für c

Das CNOT (controled not) bildet zusammen mit den 1-Qubit-Operationen einen universellen Satz von Gattern für den Einweg-Quantencomputer. Es implementiert die Funktion

$$\begin{aligned}
& \text{CNOT} \\
= \; & \left|0^{(c)}\right\rangle \left\langle 0^{(c)}\right| \otimes \mathbb{1}^{(t)} + \left|1^{(c)}\right\rangle \left\langle 1^{(c)}\right| \otimes \sigma_{\text{x}}^{(t)} \\
= \; & \left|0^{(c)}0^{(t)}\right\rangle \left\langle 0^{(c)}0^{(t)}\right| + \left|0^{(c)}1^{(t)}\right\rangle \left\langle 0^{(c)}1^{(t)}\right| + \left|1^{(c)}0^{(t)}\right\rangle \left\langle 1^{(c)}1^{(t)}\right| + \left|1^{(c)}1^{(t)}\right\rangle \left\langle 1^{(c)}0^{(t)}\right| \\
= \; & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}
\end{aligned}$$

Das CNOT-Gatter wird auch als XOR bezeichnet, da CNOT $|c,t\rangle = |c, c \oplus t\rangle$ist.

Das CNOT ist identisch mit $\text{CNOT}^\dagger$, seiner konjugiert Transponierten, die damit gleichzeitig seine Inverse ist, was zeigt, dass das CNOT unitär ist.

Die Korrelationsrelationen:

Nach den Bildern erhält man folgende X-Korrelationsrelation für Pin t

$$\sigma_{\text{x}}^{(2,1)}\ \sigma_{\text{x}}^{(4,1)}\,|\phi\rangle \quad = \quad \kappa^{(2,1)}\kappa^{(4,1)}\,|\phi\rangle$$

mit $\text{Korr}_\text{X} = \{(2,1),(4,1)\}$, $\text{XMes} = \{(3,1),(3,2),(3,3),(2,4)(3,4),(4,4)\}$, $\text{Pins} = \{(2,1),(4,1)\}$ also

$$\prod_{i\in\text{Korr}_\text{X}} \sigma_{\text{x}}^{(i)}\ |\phi\rangle \quad = \quad \prod_{i\in\text{Korr}_\text{X}} \kappa^{(i)}\,|\phi\rangle$$

Nun wendet man auf den Zustand den Messprojektor $\mathcal{P}_\text{CNOT} = \prod_{i\in\text{XMes}} \frac{1^{(i)}+\mu^{(i)}\sigma_{\text{x}}^{(i)}}{2}$ an, der den Zustand in den nach der Messung verwandelt:

$$\mathcal{P}_\text{CNOT} \prod_{i\in\text{Korr}_\text{X}} \sigma_{\text{x}}^{(i)}\ |\phi\rangle \quad = \quad \mathcal{P}_\text{CNOT} \prod_{i\in\text{Korr}_\text{X}} \kappa^{(i)}\,|\phi\rangle$$

was wegen $\text{XMeas} \cap \text{Korr}_\text{X} = \emptyset$ äquivalent zu

$$\prod_{i\in\text{Korr}_\text{X}} \sigma_{\text{x}}^{(i)}\ \mathcal{P}_\text{CNOT}\,|\phi\rangle \quad = \quad \prod_{i\in\text{Korr}_\text{X}} \kappa^{(i)}\mathcal{P}_\text{CNOT}\,|\phi\rangle$$

ist. Mit $|\phi_\text{CNOT}\rangle := \mathcal{P}_\text{CNOT}\,|\phi\rangle$ folgt dann

$$\sigma_{\text{x}}^{(\text{t})}\sigma_{\text{x}}^{(\text{out(t)})}\,|\phi_\text{CNOT}\rangle \quad = \quad \kappa^{(2,1)}\kappa^{(4,1)}\,|\phi_\text{CNOT}\rangle$$

Die Z Korrelationsrelation für Pin t ist nach dem Bild

$$\sigma_{\text{z}}^{(2,1)}\ \sigma_{\text{x}}^{(3,1)}\sigma_{\text{x}}^{(3,3)}\sigma_{\text{x}}^{(4,4)}\sigma_{\text{z}}^{(4,1)}\sigma_{\text{z}}^{(5,4)}\,|\phi\rangle \quad = \quad \kappa^{(3,1)}\kappa^{(3,3)}\kappa^{(4,4)}\,|\phi\rangle$$

mit $\underline{\text{Korr}_\text{Z}} = \{(3,1),(3,3),(4,4)\}$ ($\underline{\text{Korr}_q}$ ist die Menge der q-Korrelationszentren), $\text{XMes} = \{(3,1),(3,2),(3,3),(2,4)(3,4),(4,4)\}$, $\text{Pins} = \{(2,1),(4,1),(5,4)\}$ und $\overline{\text{Korr}_\text{Z}} = \text{XMes} - \underline{\text{Korr}_\text{Z}}$ also

$$\prod_{i\in\text{Pins}} \sigma_{\text{z}}^{(i)} \prod_{i\in\underline{\text{Korr}_\text{Z}}} \sigma_{\text{x}}^{(i)}\ |\phi\rangle \quad = \quad \prod_{i\in\underline{\text{Korr}_\text{Z}}} \kappa^{(i)}\,|\phi\rangle$$

Nun wendet man auf den Zustand den Messprojektor $\mathcal{P}_{\mathrm{CNOT}} = \prod_{i \in \mathrm{XMes}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2}$ an, der den Zustand in den nach der Messung verwandelt:

$$\mathcal{P}_{\mathrm{CNOT}} \prod_{i \in \mathrm{Pins}} \sigma_{\mathrm{z}}^{(i)} \prod_{i \in \underline{\mathrm{Korr_Z}}} \sigma_{\mathrm{x}}^{(i)} \, |\phi\rangle = \mathcal{P}_{\mathrm{CNOT}} \prod_{i \in \underline{\mathrm{Korr_Z}}} \kappa^{(i)} \, |\phi\rangle$$

$$\prod_{i \in \mathrm{Pins}} \sigma_{\mathrm{z}}^{(i)} \prod_{i \in \underline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2} \sigma_{\mathrm{x}}^{(i)} \prod_{i \in \overline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2} \, |\phi\rangle = \prod_{i \in \underline{\mathrm{Korr_Z}}} \kappa^{(i)} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle$$

$$\prod_{i \in \mathrm{Pins}} \sigma_{\mathrm{z}}^{(i)} \prod_{i \,\in\, \underline{\mathrm{Korr_Z}}} \mu^{(i)} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2} \prod_{i \in \overline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2} \, |\phi\rangle = \prod_{i \in \underline{\mathrm{Korr_Z}}} \kappa^{(i)} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle$$

$$\prod_{i \in \mathrm{Pins}} \sigma_{\mathrm{z}}^{(i)} \prod_{i \,\in\, \underline{\mathrm{Korr_Z}}} \mu^{(i)}$$

$$\prod_{i \,\in\, \underline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2} \prod_{i \in \overline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2} \, |\phi\rangle = \prod_{i \in \underline{\mathrm{Korr_Z}}} \kappa^{(i)} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle$$

$$\prod_{i \in \mathrm{Pins}} \sigma_{\mathrm{z}}^{(i)} \prod_{i \,\in\, \underline{\mathrm{Korr_Z}}} \mu^{(i)} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle = \prod_{i \in \underline{\mathrm{Korr_Z}}} \kappa^{(i)} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle$$

$$\sigma_{\mathrm{z}}^{(\mathrm{t})} \sigma_{\mathrm{z}}^{(\mathrm{c})} \sigma_{\mathrm{z}}^{(\mathrm{out(t)})} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle = \prod_{i \,\in\, \underline{\mathrm{Korr_Z}}} \mu^{(i)} \prod_{i \in \underline{\mathrm{Korr_Z}}} \kappa^{(i)} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle$$

mit $|\phi_{\mathrm{CNOT}}\rangle := \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle$ dann

$$\sigma_{\mathrm{z}}^{(\mathrm{t})} \sigma_{\mathrm{z}}^{(\mathrm{c})} \sigma_{\mathrm{z}}^{(\mathrm{out(t)})} \, |\phi_{\mathrm{CNOT}}\rangle = \prod_{i \in \underline{\mathrm{Korr_Z}}} \mu^{(i)} \kappa^{(i)} \, |\phi_{\mathrm{CNOT}}\rangle$$

Jetzt zum Eingangspin c.
Für es gilt nach dem Diagramm für die X-Korrelationen

$$\sigma_{\mathrm{x}}^{(2,1)} \, \sigma_{\mathrm{x}}^{(3,2)} \sigma_{\mathrm{x}}^{(3,4)} \, \sigma_{\mathrm{x}}^{(1,4)} \sigma_{\mathrm{x}}^{(5,4)} \, |\phi\rangle = \kappa^{(2,1)} \kappa^{(3,2)} \kappa^{(3,4)} \kappa^{(1,4)} \kappa^{(5,4)} \, |\phi\rangle$$

mit $\mathrm{Korr_X} = \{(2,1),(3,2),(3,4),(1,4),(5,4)\}$, $\mathrm{XMes} = \{(3,1),(3,2),(3,3),(2,4)(3,4),(4,4)\}$, $\mathrm{Pins} = \{(2,1),(1,4),(5,4)\}$, $\underline{\mathrm{Korr_X}} = \underline{\mathrm{Korr_X}} \cap \mathrm{XMes}$ (die auf die in X gemessene Koordinaten beschränkte Menge) und $\overline{\mathrm{Korr_X}} = \mathrm{XMes} - \underline{\mathrm{Korr_X}}$ also

$$\prod_{i \in \mathrm{Korr_X}} \sigma_{\mathrm{x}}^{(i)} \, |\phi\rangle = \prod_{i \in \mathrm{Korr_X}} \kappa^{(i)} \, |\phi\rangle$$

Nun wendet man auf den Zustand den Messprojektor $\mathcal{P}_{\mathrm{CNOT}} = \prod_{i \in \mathrm{XMes}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2}$ an, der den Zustand in den nach der Messung verwandelt:

$$\mathcal{P}_{\mathrm{CNOT}} \prod_{i \in \mathrm{Pins}} \sigma_{\mathrm{x}}^{(i)} \prod_{i \in \underline{\mathrm{Korr_X}}} \sigma_{\mathrm{x}}^{(i)} \, |\phi\rangle = \mathcal{P}_{\mathrm{CNOT}} \prod_{i \in \mathrm{Korr_X}} \kappa^{(i)} \, |\phi\rangle$$

$$\prod_{i \in \mathrm{Pins}} \sigma_{\mathrm{x}}^{(i)} \prod_{i \in \underline{\mathrm{Korr_X}}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2} \sigma_{\mathrm{x}}^{(i)} \prod_{i \in \overline{\mathrm{Korr_X}}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2} \, |\phi\rangle = \prod_{i \in \mathrm{Korr_X}} \kappa^{(i)} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle$$

$$\prod_{i \in \mathrm{Pins}} \sigma_{\mathrm{x}}^{(i)} \prod_{i \,\in\, \underline{\mathrm{Korr_X}}} \mu^{(i)} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2} \prod_{i \in \overline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2} \, |\phi\rangle = \prod_{i \in \mathrm{Korr_X}} \kappa^{(i)} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle$$

$$\prod_{i \in \mathrm{Pins}} \sigma_{\mathrm{x}}^{(i)} \prod_{i \,\in\, \underline{\mathrm{Korr_X}}} \mu^{(i)}$$

$$\prod_{i \,\in\, \underline{\mathrm{Korr_X}}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2} \prod_{i \in \overline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2} \, |\phi\rangle = \prod_{i \in \mathrm{Korr_X}} \kappa^{(i)} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle$$

$$\prod_{i \in \mathrm{Pins}} \sigma_{\mathrm{x}}^{(i)} \prod_{i \,\in\, \underline{\mathrm{Korr_Z}}} \mu^{(i)} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle = \prod_{i \in \mathrm{Korr_X}} \kappa^{(i)} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle$$

$$\sigma_{\mathrm{x}}^{(\mathrm{c})} \sigma_{\mathrm{x}}^{(\mathrm{out(c)})} \sigma_{\mathrm{x}}^{(\mathrm{out(t)})} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle = \prod_{i \,\in\, \underline{\mathrm{Korr_X}}} \mu^{(i)} \prod_{i \in \mathrm{Korr_X}} \kappa^{(i)} \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle$$

mit $|\phi_{\mathrm{CNOT}}\rangle := \mathcal{P}_{\mathrm{CNOT}} \, |\phi\rangle$ dann

$$\sigma_{\mathrm{x}}^{(\mathrm{c})} \sigma_{\mathrm{x}}^{(\mathrm{out(c)})} \sigma_{\mathrm{x}}^{(\mathrm{out(t)})} \, |\phi_{\mathrm{CNOT}}\rangle = \prod_{i \,\in\, \underline{\mathrm{Korr_X}}} \mu^{(i)} \prod_{i \in \mathrm{Korr_X}} \kappa^{(i)} \, |\phi_{\mathrm{CNOT}}\rangle$$

$$\sigma_{\mathrm{x}}^{(5,4)} \sigma_{\mathrm{x}}^{(1,4)} \sigma_{\mathrm{x}}^{(2,1)} \, |\phi_{\mathrm{CNOT}}\rangle = \mu^{(3,2)} \mu^{(3,4)} \kappa^{(2,1)} \kappa^{(3,2)} \kappa^{(3,4)} \kappa^{(1,4)} \kappa^{(5,4)} \, |\phi_{\mathrm{CNOT}}\rangle$$

Die Z-Korrelation von c ist nach obigem Bild

$$\sigma_{\mathrm{z}}^{(1,4)} \; \sigma_{\mathrm{x}}^{(2,4)} \sigma_{\mathrm{x}}^{(4,4)} \sigma_{\mathrm{z}}^{(5,4)} \, |\phi\rangle = \kappa^{(2,4)} \kappa^{(4,4)} \, |\phi\rangle$$

mit $\underline{\mathrm{Korr_Z}} = \{(2,4),(4,4)\}$, $\mathrm{XMes} = \{(3,1),(3,2),(3,3),(2,4)(3,4),(4,4)\}$, $\mathrm{Pins} = \{(1,4),(5,4)\}$ und $\overline{\mathrm{Korr_Z}} = \mathrm{XMes} - \underline{\mathrm{Korr_Z}}$ also

$$\prod_{i \in \mathrm{Pins}} \sigma_{\mathrm{z}}^{(i)} \prod_{i \in \underline{\mathrm{Korr_Z}}} \sigma_{\mathrm{x}}^{(i)} \; |\phi\rangle = \prod_{i \in \underline{\mathrm{Korr_Z}}} \kappa^{(i)} \, |\phi\rangle$$

Nun wendet man auf den Zustand den Messprojektor $\mathcal{P}_{\mathrm{CNOT}} = \prod_{i \in \mathrm{XMes}} \frac{\mathbb{1}^{(i)} + \mu^{(i)} \sigma_{\mathrm{x}}^{(i)}}{2}$

an, der den Zustand in den nach der Messung verwandelt:

$$\mathcal{P}_{\mathrm{CNOT}} \prod_{i\in\mathrm{Pins}} \sigma_{\mathrm{z}}^{(i)} \prod_{i\in\underline{\mathrm{Korr_Z}}} \sigma_{\mathrm{x}}^{(i)} \left|\phi\right\rangle = \mathcal{P}_{\mathrm{CNOT}} \prod_{i\in\underline{\mathrm{Korr_Z}}} \kappa^{(i)} \left|\phi\right\rangle$$

$$\prod_{i\in\mathrm{Pins}} \sigma_{\mathrm{z}}^{(i)} \prod_{i\in\underline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2}\sigma_{\mathrm{x}}^{(i)} \prod_{i\in\overline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2} \left|\phi\right\rangle = \prod_{i\in\underline{\mathrm{Korr_Z}}} \kappa^{(i)}\mathcal{P}_{\mathrm{CNOT}} \left|\phi\right\rangle$$

$$\prod_{i\in\mathrm{Pins}} \sigma_{\mathrm{z}}^{(i)} \prod_{i\,\in\,\underline{\mathrm{Korr_Z}}} \mu^{(i)} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2} \prod_{i\in\overline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2} \left|\phi\right\rangle = \prod_{i\in\underline{\mathrm{Korr_Z}}} \kappa^{(i)}\mathcal{P}_{\mathrm{CNOT}} \left|\phi\right\rangle$$

$$\prod_{i\in\mathrm{Pins}} \sigma_{\mathrm{z}}^{(i)} \prod_{i\,\in\,\underline{\mathrm{Korr_Z}}} \mu^{(i)}$$

$$\prod_{i\,\in\,\underline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2} \prod_{i\in\overline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2} \left|\phi\right\rangle = \prod_{i\in\underline{\mathrm{Korr_Z}}} \kappa^{(i)}\mathcal{P}_{\mathrm{CNOT}} \left|\phi\right\rangle$$

$$\prod_{i\in\mathrm{Pins}} \sigma_{\mathrm{z}}^{(i)} \prod_{i\,\in\,\underline{\mathrm{Korr_Z}}} \mu^{(i)}\mathcal{P}_{\mathrm{CNOT}} \left|\phi\right\rangle = \prod_{i\in\underline{\mathrm{Korr_Z}}} \kappa^{(i)}\mathcal{P}_{\mathrm{CNOT}} \left|\phi\right\rangle$$

$$\sigma_{\mathrm{z}}^{(\mathrm{c})}\sigma_{\mathrm{z}}^{(\mathrm{out(c)})}\mathcal{P}_{\mathrm{CNOT}} \left|\phi\right\rangle = \prod_{i\,\in\,\underline{\mathrm{Korr_Z}}} \mu^{(i)} \prod_{i\in\underline{\mathrm{Korr_Z}}} \kappa^{(i)}\mathcal{P}_{\mathrm{CNOT}} \left|\phi\right\rangle$$

mit $\left|\phi_{\mathrm{CNOT}}\right\rangle := \mathcal{P}_{\mathrm{CNOT}} \left|\phi\right\rangle$ dann

$$\sigma_{\mathrm{z}}^{(\mathrm{c})}\sigma_{\mathrm{z}}^{(\mathrm{out(c)})} \left|\phi_{\mathrm{CNOT}}\right\rangle = \prod_{i\in\underline{\mathrm{Korr_Z}}} \mu^{(i)}\kappa^{(i)} \left|\phi_{\mathrm{CNOT}}\right\rangle$$

Sei $U_{\mathrm{CNOT}} = \left|0^{(c)}\right\rangle\left\langle 0^{(c)}\right| \otimes \mathbb{1}^{(t)} + \left|1^{(c)}\right\rangle\left\langle 1^{(c)}\right| \otimes \sigma_{\mathrm{x}}^{(t)}$. Wie oben gezeigt, folgt aus den Korrelationsrelationen

$$\sigma_{\mathrm{x}}^{(\mathrm{c})}\sigma_{\mathrm{x}}^{(\mathrm{out(c)})}\sigma_{\mathrm{x}}^{(\mathrm{out(t)})} \left|\phi_{\mathrm{CNOT}}\right\rangle = \prod_{i\,\in\,\underline{\mathrm{Korr_X}}} \mu^{(i)} \prod_{i\in\mathrm{Korr_X}} \kappa^{(i)} \left|\phi_{\mathrm{CNOT}}\right\rangle$$

Mit $AA^{\dagger} = \mathbb{1}$ gilt dann

$$\sigma_{\mathrm{x}}^{(\mathrm{c})}U_{\mathrm{CNOT}}^{\mathrm{out(in)}}U_{\mathrm{CNOT}}^{\dagger\mathrm{out(in)}}\sigma_{\mathrm{x}}^{(\mathrm{out(c)})}\sigma_{\mathrm{x}}^{(\mathrm{out(t)})} \left|\phi_{\mathrm{CNOT}}\right\rangle = \prod_{i\,\in\,\underline{\mathrm{Korr_X}}} \mu^{(i)} \prod_{i\in\mathrm{Korr_X}} \kappa^{(i)} \left|\phi_{\mathrm{CNOT}}\right\rangle$$

Da

$$
\begin{aligned}
& U_{\mathrm{CNOT}}^{\dagger \mathrm{out(in)}} \sigma_{\mathrm{x}}^{(\mathrm{out(c)})} \sigma_{\mathrm{x}}^{(\mathrm{out(t)})} \\
= \; & (\left|0^{(\mathrm{out(c)})}\right\rangle \left\langle 0^{(\mathrm{out(c)})}\right| \otimes \mathbb{1}^{\mathrm{out(t)}} + \left|1^{(\mathrm{out(c)})}\right\rangle \left\langle 1^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{x}}^{\mathrm{out(t)}}) \sigma_{\mathrm{x}}^{(\mathrm{out(c)})} \sigma_{\mathrm{x}}^{(\mathrm{out(t)})} \\
= \; & (\left|0^{(\mathrm{out(c)})}\right\rangle \left\langle 1^{(\mathrm{out(c)})}\right| \otimes \mathbb{1}^{\mathrm{out(t)}} + \left|1^{(\mathrm{out(c)})}\right\rangle \left\langle 0^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{x}}^{\mathrm{out(t)}}) \sigma_{\mathrm{x}}^{(\mathrm{out(t)})} \\
= \; & \left|0^{(\mathrm{out(c)})}\right\rangle \left\langle 1^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{x}}^{\mathrm{out(t)}} + \left|1^{(\mathrm{out(c)})}\right\rangle \left\langle 0^{(\mathrm{out(c)})}\right| \otimes \mathbb{1}^{\mathrm{out(t)}} \\
= \; & \left|1^{(\mathrm{out(c)})}\right\rangle \left\langle 0^{(\mathrm{out(c)})}\right| \otimes \mathbb{1}^{\mathrm{out(t)}} + \left|0^{(\mathrm{out(c)})}\right\rangle \left\langle 1^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{x}}^{\mathrm{out(t)}} \\
= \; & (\sigma_{\mathrm{x}}^{(\mathrm{out(c)})} \left|0^{(\mathrm{out(c)})}\right\rangle) \left\langle 0^{(\mathrm{out(c)})}\right| \otimes \mathbb{1}^{\mathrm{out(t)}} + (\sigma_{\mathrm{x}}^{(\mathrm{out(c)})} \left|1^{(\mathrm{out(c)})}\right\rangle) \left\langle 1^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{x}}^{\mathrm{out(t)}} \\
= \; & \sigma_{\mathrm{x}}^{(\mathrm{out(c)})} (\left|0^{(\mathrm{out(c)})}\right\rangle \left\langle 0^{(\mathrm{out(c)})}\right| \otimes \mathbb{1}^{\mathrm{out(t)}} + \left|1^{(\mathrm{out(c)})}\right\rangle \left\langle 1^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{x}}^{\mathrm{out(t)}}) \\
= \; & \sigma_{\mathrm{x}}^{(\mathrm{out(c)})} U_{\mathrm{CNOT}}^{\dagger \mathrm{out(in)}}
\end{aligned}
$$

gilt

$$
\sigma_{\mathrm{x}}^{(\mathrm{c})} U_{\mathrm{CNOT}}^{\mathrm{out(in)}} \sigma_{\mathrm{x}}^{(\mathrm{out(c)})} U_{\mathrm{CNOT}}^{\dagger \mathrm{out(in)}} \left|\phi_{\mathrm{CNOT}}\right\rangle = \prod_{i \in \underline{\mathrm{Korr_X}}} \mu^{(i)} \prod_{i \in \mathrm{Korr_X}} \kappa^{(i)} \left|\phi_{\mathrm{CNOT}}\right\rangle
$$

Genauso wird aus

$$
\sigma_{\mathrm{z}}^{(\mathrm{t})} U_{\mathrm{CNOT}}^{\mathrm{out(in)}} U_{\mathrm{CNOT}}^{\dagger \mathrm{out(in)}} \sigma_{\mathrm{z}}^{(\mathrm{c})} \sigma_{\mathrm{z}}^{(\mathrm{out(t)})} \left|\phi_{\mathrm{CNOT}}\right\rangle = \prod_{i \in \underline{\mathrm{Korr_Z}}} \mu^{(i)} \kappa^{(i)} \left|\phi_{\mathrm{CNOT}}\right\rangle
$$

mit

$$
\begin{aligned}
& U_{\mathrm{CNOT}}^{\dagger \mathrm{out(in)}} \sigma_{\mathrm{z}}^{(\mathrm{c})} \sigma_{\mathrm{z}}^{(\mathrm{out(t)})} \\
= \; & (\left|0^{(\mathrm{out(c)})}\right\rangle \left\langle 0^{(\mathrm{out(c)})}\right| \otimes \mathbb{1}^{\mathrm{out(t)}} + \left|1^{(\mathrm{out(c)})}\right\rangle \left\langle 1^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{x}}^{\mathrm{out(t)}}) \sigma_{\mathrm{z}}^{(\mathrm{c})} \sigma_{\mathrm{z}}^{(\mathrm{out(t)})} \\
= \; & (\left|0^{(\mathrm{out(c)})}\right\rangle \left\langle 0^{(\mathrm{out(c)})}\right| \otimes \mathbb{1}^{\mathrm{out(t)}} - \left|1^{(\mathrm{out(c)})}\right\rangle \left\langle 1^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{x}}^{\mathrm{out(t)}}) \sigma_{\mathrm{z}}^{(\mathrm{out(t)})} \\
= \; & (\left|0^{(\mathrm{out(c)})}\right\rangle \left\langle 0^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{z}}^{(\mathrm{out(t)})} - \left|1^{(\mathrm{out(c)})}\right\rangle \left\langle 1^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{x}}^{\mathrm{out(t)}} \sigma_{\mathrm{z}}^{(\mathrm{out(t)})}) \\
= \; & (\left|0^{(\mathrm{out(c)})}\right\rangle \left\langle 0^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{z}}^{(\mathrm{out(t)})} - \left|1^{(\mathrm{out(c)})}\right\rangle \left\langle 1^{(\mathrm{out(c)})}\right| \otimes -\sigma_{\mathrm{z}}^{(\mathrm{out(t)})} \sigma_{\mathrm{x}}^{\mathrm{out(t)}}) \\
= \; & (\left|0^{(\mathrm{out(c)})}\right\rangle \left\langle 0^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{z}}^{(\mathrm{out(t)})} + \left|1^{(\mathrm{out(c)})}\right\rangle \left\langle 1^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{z}}^{(\mathrm{out(t)})} \sigma_{\mathrm{x}}^{\mathrm{out(t)}}) \\
= \; & \sigma_{\mathrm{z}}^{(\mathrm{out(t)})} (\left|0^{(\mathrm{out(c)})}\right\rangle \left\langle 0^{(\mathrm{out(c)})}\right| \otimes \mathbb{1}^{\mathrm{out(t)}} + \left|1^{(\mathrm{out(c)})}\right\rangle \left\langle 1^{(\mathrm{out(c)})}\right| \otimes \sigma_{\mathrm{x}}^{\mathrm{out(t)}})
\end{aligned}
$$

die passende $\sigma_{\mathrm{z}}^{(\mathrm{t})}$-Eigenwertgleichung

$$
\sigma_{\mathrm{z}}^{(\mathrm{t})} U_{\mathrm{CNOT}}^{\mathrm{out(in)}} \sigma_{\mathrm{z}}^{(\mathrm{out(t)})} U_{\mathrm{CNOT}}^{\dagger \mathrm{out(in)}} \left|\phi_{\mathrm{CNOT}}\right\rangle = \prod_{i \in \underline{\mathrm{Korr_Z}}} \mu^{(i)} \kappa^{(i)} \left|\phi_{\mathrm{CNOT}}\right\rangle
$$

Dann gilt nach obigen Herleitungen (also den beiden zum Schluss umgeformten und den beiden sich direkt aus den Korrelationsrelationen ergebenden) und mit $\text{in} = \{c, t\} = \{(5,4),(4,1)\}$, $\text{out}((5,4)) = (1,4)$ und $\text{out}((4,1)) = (2,1)$ das gewünschte Ergebnis

$$\forall i \in \text{in}. \sigma_{\text{x}}^{(i)} \text{CNOT}^{(\text{out}(\text{in}))} \sigma_{\text{x}}^{(\text{out}(i))} \text{CNOT}^{\dagger(\text{out}(\text{in}))} \left|\phi_{\text{CNOT}}\right\rangle = \tau_{\text{x}}^{(i)} \left|\phi_{\text{CNOT}}\right\rangle$$

und

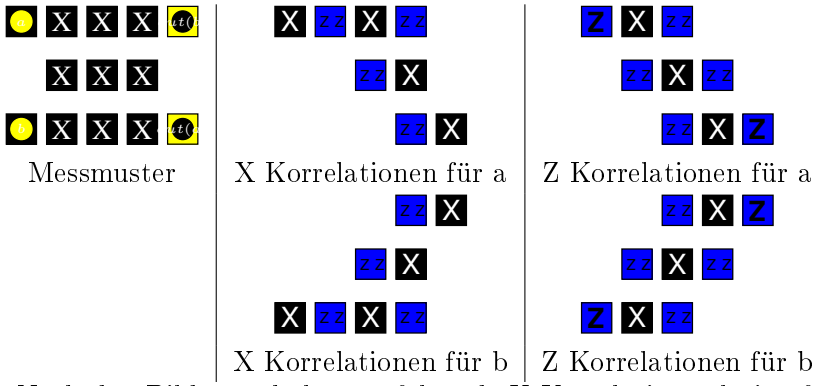
$$\forall i \in \text{in}. \sigma_{\text{z}}^{(i)} \text{CNOT}^{(\text{out}(\text{in}))} \sigma_{\text{z}}^{(\text{out}(i))} \text{CNOT}^{\dagger(\text{out}(\text{in}))} \left|\phi_{\text{CNOT}}\right\rangle = \tau_{\text{z}}^{(i)} \left|\phi_{\text{CNOT}}\right\rangle$$

mit

$$\begin{aligned}
\tau_{\text{x}}^{(\text{c})} &= \prod_{i \in \underline{\text{Korr}_{\text{X}}^{\text{c}}}} \mu^{(i)} \prod_{i \in \text{Korr}_{\text{X}}^{\text{c}}} \kappa^{(i)} &&= \mu^{(3,2)} \mu^{(3,4)} \kappa^{(2,1)} \kappa^{(3,2)} \kappa^{(1,4)} \kappa^{(3,4)} \kappa^{(5,4)} \\
\tau_{\text{z}}^{(\text{c})} &= \prod_{i \in \underline{\text{Korr}_{\text{Z}}^{\text{c}}}} \mu^{(i)} \kappa^{(i)} &&= \mu^{(2,4)} \mu^{(4,4)} \kappa^{(2,4)} \kappa^{(4,4)} \\
\tau_{\text{x}}^{(\text{t})} &= \prod_{i \in \underline{\text{Korr}_{\text{X}}^{\text{t}}}} \mu^{(i)} \prod_{i \in \text{Korr}_{\text{X}}^{\text{t}}} \kappa^{(i)} &&= \kappa^{(2,1)} \kappa^{(4,1)} \\
\tau_{\text{z}}^{(\text{t})} &= \prod_{i \in \underline{\text{Korr}_{\text{Z}}^{\text{t}}}} \mu^{(i)} \kappa^{(i)} &&= \mu^{(3,1)} \mu^{(3,3)} \mu^{(4,4)} \kappa^{(3,1)} \kappa^{(3,3)} \kappa^{(4,4)}
\end{aligned}$$

.

4.2.3 Swap



Nach den Bildern erhält man folgende X Korrelationsrelation für Pin a

$$\sigma_{\text{x}}^{(1,1)} \sigma_{\text{x}}^{(3,1)} \sigma_{\text{x}}^{(4,2)} \sigma_{\text{x}}^{(5,3)} \left|\phi\right\rangle = \kappa^{(1,1)} \kappa^{(3,1)} \kappa^{(4,2)} \kappa^{(5,3)} \left|\phi\right\rangle$$

mit $\text{Korr}_{\text{X}} = \{(1,1),(3,1),(4,2),(5,3)\}$, $\text{Pins} = \{(1,1),(5,3)\}$, $\underline{\text{Korr}_{\text{X}}} = \text{Korr}_{\text{X}} - \text{Pins}$ (also die auf die in X gemessene Koordinaten beschränkte Menge) und $\overline{\text{Korr}_{\text{X}}} = \{2,3,4\} \times \{1,2,3\} - \underline{\text{Korr}_{\text{X}}}$ also

$$\prod_{i \in \text{Korr}_{\text{X}}} \sigma_{\text{x}}^{(i)} \left|\phi\right\rangle = \prod_{i \in \text{Korr}_{\text{X}}} \kappa^{(i)} \left|\phi\right\rangle$$

Nun wendet man auf den Zustand den Messprojektor $\mathcal{P}_{\text{Swap}} = \prod_{i\in\{2,3,4\}\times\{1,2,3\}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\text{x}}^{(i)}}{2}$ an, der den Zustand in den nach der Messung verwandelt:

$$\mathcal{P}_{\text{Swap}} \prod_{i\in\text{Korr}_{\text{X}}} \sigma_{\text{x}}^{(i)} \,|\phi\rangle = \mathcal{P}_{\text{Swap}} \prod_{i\in\text{Korr}_{\text{X}}} \kappa^{(i)} \,|\phi\rangle$$

$$\sigma_{\text{x}}^{(\text{a})}\sigma_{\text{x}}^{(\text{out(a)})} \prod_{i\,\in\,\underline{\text{Korr}_{\text{X}}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\text{x}}^{(i)}}{2}\sigma_{\text{x}}^{(i)} \prod_{\overline{\text{Korr}_{\text{X}}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\text{x}}^{(i)}}{2} \,|\phi\rangle = \prod_{i\in\text{Korr}_{\text{X}}} \kappa^{(i)}\mathcal{P}_{\text{Swap}} \,|\phi\rangle$$

$$\sigma_{\text{x}}^{(\text{a})}\sigma_{\text{x}}^{(\text{out(a)})} \prod_{i\,\in\,\underline{\text{Korr}_{\text{X}}}} \mu^{(i)}\frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\text{x}}^{(i)}}{2} \prod_{\overline{\text{Korr}_{\text{X}}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\text{x}}^{(i)}}{2} \,|\phi\rangle = \prod_{i\in\text{Korr}_{\text{X}}} \kappa^{(i)}\mathcal{P}_{\text{Swap}} \,|\phi\rangle$$

$$\sigma_{\text{x}}^{(\text{a})}\sigma_{\text{x}}^{(\text{out(a)})} \prod_{i\,\in\,\underline{\text{Korr}_{\text{X}}}} \mu^{(i)} \prod_{i\,\in\,\underline{\text{Korr}_{\text{X}}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\text{x}}^{(i)}}{2} \prod_{\overline{\text{Korr}_{\text{X}}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\text{x}}^{(i)}}{2} \,|\phi\rangle = \prod_{i\in\text{Korr}_{\text{X}}} \kappa^{(i)}\mathcal{P}_{\text{Swap}} \,|\phi\rangle$$

$$\sigma_{\text{x}}^{(\text{a})}\sigma_{\text{x}}^{(\text{out(a)})} \prod_{i\,\in\,\underline{\text{Korr}_{\text{X}}}} \mu^{(i)}\mathcal{P}_{\text{Swap}} \,|\phi\rangle = \prod_{i\in\text{Korr}_{\text{X}}} \kappa^{(i)}\mathcal{P}_{\text{Swap}} \,|\phi\rangle$$

$$\sigma_{\text{x}}^{(\text{a})}\sigma_{\text{x}}^{(\text{out(a)})}\mathcal{P}_{\text{Swap}} \,|\phi\rangle = \prod_{i\,\in\,\underline{\text{Korr}_{\text{X}}}} \mu^{(i)} \prod_{i\in\text{Korr}_{\text{X}}} \kappa^{(i)}\mathcal{P}_{\text{Swap}} \,|\phi\rangle$$

mit $|\phi_{\text{Swap}}\rangle := \mathcal{P}_{\text{Swap}} \,|\phi\rangle$ dann

$$\sigma_{\text{x}}^{(\text{a})}\sigma_{\text{x}}^{(\text{out(a)})} \,|\phi_{\text{Swap}}\rangle = \prod_{i\,\in\,\underline{\text{Korr}_{\text{X}}}} \mu^{(i)} \prod_{i\in\text{Korr}_{\text{X}}} \kappa^{(i)} \,|\phi_{\text{Swap}}\rangle$$

Die Z Korrelationsrelation für Pin a ist nach dem Bild

$$\sigma_{\text{z}}^{(1,1)}\ \sigma_{\text{x}}^{(2,1)}\sigma_{\text{x}}^{(3,2)}\sigma_{\text{x}}^{(4,3)}\sigma_{\text{z}}^{(5,3)} \,|\phi\rangle = \kappa^{(2,1)}\kappa^{(3,2)}\kappa^{(4,3)} \,|\phi\rangle$$

mit $\underline{\text{Korr}_{\text{Z}}} = \{(2,1),(3,2),(4,3)\}$, $\text{Pins} = \{(1,1),(5,3)\}$ und $\overline{\text{Korr}_{\text{Z}}} = \{2,3,4\} \times \{1,2,3\} - \underline{\text{Korr}_{\text{Z}}}$ also

$$\sigma_{\text{z}}^{(1,1)}\sigma_{\text{z}}^{(5,3)} \prod_{i\in\underline{\text{Korr}_{\text{Z}}}} \sigma_{\text{x}}^{(i)} \,|\phi\rangle = \prod_{i\in\underline{\text{Korr}_{\text{Z}}}} \kappa^{(i)} \,|\phi\rangle$$

Nun wendet man auf den Zustand den Messprojektor $\mathcal{P}_{\text{Swap}} =$

$\prod_{i\in\{2,3,4\}\times\{1,2,3\}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2}$ an, der den Zustand in den nach der Messung verwandelt:

$$\begin{aligned}
\mathcal{P}_{\mathrm{Swap}}\sigma_{\mathrm{z}}^{(1,1)}\sigma_{\mathrm{z}}^{(5,3)} \prod_{i\in\underline{\mathrm{Korr_Z}}} \sigma_{\mathrm{x}}^{(i)} \left|\phi\right\rangle &= \mathcal{P}_{\mathrm{Swap}} \prod_{i\in\underline{\mathrm{Korr_Z}}} \kappa^{(i)} \left|\phi\right\rangle \\
\sigma_{\mathrm{z}}^{(1,1)}\sigma_{\mathrm{z}}^{(5,3)} \prod_{i\in\underline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2}\sigma_{\mathrm{x}}^{(i)} \prod_{i\in\overline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2} \left|\phi\right\rangle &= \prod_{i\in\underline{\mathrm{Korr_Z}}} \kappa^{(i)}\mathcal{P}_{\mathrm{Swap}} \left|\phi\right\rangle \\
\sigma_{\mathrm{z}}^{(1,1)}\sigma_{\mathrm{z}}^{(5,3)} \prod_{i\,\in\,\underline{\mathrm{Korr_Z}}} \mu^{(i)}\frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2} \prod_{i\in\overline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2} \left|\phi\right\rangle &= \prod_{i\in\underline{\mathrm{Korr_Z}}} \kappa^{(i)}\mathcal{P}_{\mathrm{Swap}} \left|\phi\right\rangle \\
\sigma_{\mathrm{z}}^{(1,1)}\sigma_{\mathrm{z}}^{(5,3)} \prod_{i\,\in\,\underline{\mathrm{Korr_Z}}} \mu^{(i)} \prod_{i\,\in\,\underline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2} \prod_{i\in\overline{\mathrm{Korr_Z}}} \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2} \left|\phi\right\rangle &= \prod_{i\in\underline{\mathrm{Korr_Z}}} \kappa^{(i)}\mathcal{P}_{\mathrm{Swap}} \left|\phi\right\rangle \\
\sigma_{\mathrm{z}}^{(1,1)}\sigma_{\mathrm{z}}^{(5,3)} \prod_{i\,\in\,\underline{\mathrm{Korr_Z}}} \mu^{(i)}\mathcal{P}_{\mathrm{Swap}} \left|\phi\right\rangle &= \prod_{i\in\underline{\mathrm{Korr_Z}}} \kappa^{(i)}\mathcal{P}_{\mathrm{Swap}} \left|\phi\right\rangle \\
\sigma_{\mathrm{z}}^{(\mathrm{a})}\sigma_{\mathrm{z}}^{(\mathrm{out(a)})}\mathcal{P}_{\mathrm{Swap}} \left|\phi\right\rangle = \prod_{i\,\in\,\underline{\mathrm{Korr_Z}}} \mu^{(i)} \prod_{i\in\underline{\mathrm{Korr_Z}}} \kappa^{(i)}\mathcal{P}_{\mathrm{Swap}} \left|\phi\right\rangle &
\end{aligned}$$

mit $\left|\phi_{\mathrm{Swap}}\right\rangle := \mathcal{P}_{\mathrm{Swap}} \left|\phi\right\rangle$ dann

$$\sigma_{\mathrm{z}}^{(\mathrm{a})}\sigma_{\mathrm{z}}^{(\mathrm{out(a)})} \left|\phi_{\mathrm{Swap}}\right\rangle = \prod_{i\in\underline{\mathrm{Korr_Z}}} \mu^{(i)}\kappa^{(i)} \left|\phi_{\mathrm{Swap}}\right\rangle$$

Für Pin b werden analog die Gleichungen

$$\sigma_{\mathrm{z}}^{(\mathrm{b})}\sigma_{\mathrm{z}}^{(\mathrm{out(b)})} \left|\phi_{\mathrm{Swap}}\right\rangle = \prod_{i\in\underline{\mathrm{Korr_Z}}} \mu^{(i)}\kappa^{(i)} \left|\phi_{\mathrm{Swap}}\right\rangle$$

$$\sigma_{\mathrm{x}}^{(\mathrm{b})}\sigma_{\mathrm{x}}^{(\mathrm{out(b)})} \left|\phi_{\mathrm{Swap}}\right\rangle = \prod_{i\,\in\,\underline{\mathrm{Korr_X}}} \mu^{(i)} \prod_{i\in\mathrm{Korr_X}} \kappa^{(i)} \left|\phi_{\mathrm{Swap}}\right\rangle$$

mit $\underline{\mathrm{Korr_Z}} = \{(2,3),(3,2),(4,1)\}$, $\mathrm{Korr_X} = \{(1,3),(3,3),(4,2),(5,1)\}$ (siehe obige Diagramme) und $\underline{\mathrm{Korr_X}} = \{(3,3),(4,2)\}$ abgeleitet.

Sei $U_{\mathrm{Swap}} = \mathbb{1} \otimes \mathbb{1}$. Dann gilt nach obigen Herleitungen und mit in $= \{\mathrm{a},\mathrm{b}\} = \{(1,1),(1,3)\}$, $\mathrm{out}((1,3)) = (5,1)$ und $\mathrm{out}((1,1)) = (5,3)$ erhalte ich dann das gewünschte Ergebnis

$$\forall i \in \text{in}. \sigma_{\text{x}}^{(i)} U_{\text{Swap}}^{\text{out(in)}} \sigma_{\text{x}}^{\text{out}(i)} U_{\text{Swap}}^{\dagger\text{out(in)}} |\phi_{\text{Swap}}\rangle = \tau_{\text{x}}^{(i)} |\phi_{\text{Swap}}\rangle$$

und

$$\forall i \in \text{in}. \sigma_{\text{z}}^{(i)} U_{\text{Swap}}^{\text{out(in)}} \sigma_{\text{z}}^{\text{out}(i)} U_{\text{Swap}}^{\dagger\text{out(in)}} |\phi_{\text{Swap}}\rangle = \tau_{\text{z}}^{(i)} |\phi_{\text{Swap}}\rangle$$

mit

$$\begin{aligned}
\tau_{\text{x}}^{(\text{a})} &= \prod_{i \in \underline{\text{Korr}_{\text{X}}^{\text{a}}}} \mu^{(i)} \prod_{i \in \text{Korr}_{\text{X}}^{\text{a}}} \kappa^{(i)} &&= \mu^{(3,1)} \mu^{(4,2)} \kappa^{(1,1)} \kappa^{(3,1)} \kappa^{(4,2)} \kappa^{(5,3)} \\
\tau_{\text{z}}^{(\text{a})} &= \prod_{i \in \underline{\text{Korr}_{\text{Z}}^{\text{a}}}} \mu^{(i)} \kappa^{(i)} &&= \mu^{(2,1)} \mu^{(3,2)} \mu^{(4,3)} \kappa^{(2,1)} \kappa^{(3,2)} \kappa^{(4,3)} \\
\tau_{\text{x}}^{(\text{b})} &= \prod_{i \in \underline{\text{Korr}_{\text{X}}^{\text{b}}}} \mu^{(i)} \prod_{i \in \text{Korr}_{\text{X}}^{\text{b}}} \kappa^{(i)} &&= \mu^{(3,3)} \mu^{(4,2)} \kappa^{(1,3)} \kappa^{(3,3)} \kappa^{(4,2)} \kappa^{(5,1)} \\
\tau_{\text{z}}^{(\text{b})} &= \prod_{i \in \underline{\text{Korr}_{\text{Z}}^{\text{b}}}} \mu^{(i)} \kappa^{(i)} &&= \mu^{(2,3)} \mu^{(3,2)} \mu^{(4,1)} \kappa^{(2,3)} \kappa^{(3,2)} \kappa^{(4,1)}
\end{aligned}$$

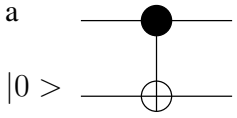
.

4.2.4 $|0\rangle$ und $|1\rangle$

Eine feste 0 an einem Eingang wird realisiert, indem man diesen in der Z-Basis misst. Das zufällige Messergebnis legt fest, ob eine 0 oder 1 am Eingang anliegt. Hat man den ungewollten Eigenwert gemessen und damit das Qubit in den falschen Eigenvektor projiziert, muss man, wie bei den anderen zufälligen Messergebnissen auch, das als Beiprodukt bei der weiteren Verarbeitung berücksichtigen.

4.2.5 Fan-Out

Das Fan-Out ist eine Umsetzung folgenden Schaltkreises:



Das Eingangspin am Targetqubit eines CNOT wird in Z gemessen.

4.2.6 Hadamard

Messmuster | X Korrelationen | Z Korrelationen

Die Funktion des Hadamard ist $\left|+^{(a)}\right\rangle \left\langle 0^{(a)}\right| + \left|-^{(a)}\right\rangle \left\langle 1^{(a)}\right|$. Man kann es als auf ein Qubit reduzierte Quantenfouriertransformation betrachten. Es wird zum einen oft dazu verwendet, den in vielen Qantenalgorithmen als Ausgangszustand vorkommenden $\bigotimes_{i=1}^{n} |+\rangle$ zu präparieren, zum anderen vertauscht es in Berechnungen σ_{x}- mit σ_{z}-Drehungen, was hier z.B. bei der Konstruktion des Toffoli benutzt wird.

Die Korrelationsrelationen zum Beleg der X-Vorraussetzung sind

$$\sigma_{\text{x}}^{(1,1)} \sigma_{\text{x}}^{(3,1)} \sigma_{\text{z}}^{(4,1)} \sigma_{\text{z}}^{(3,1)} \sigma_{\text{x}}^{(4,1)} \sigma_{\text{z}}^{(5,1)} |\phi\rangle = \kappa^{(1,1)} \kappa^{(3,1)} \kappa^{(4,1)} |\phi\rangle$$

mit $\sigma_\mathrm{x}\sigma_\mathrm{z} = -\sigma_\mathrm{z}\sigma_\mathrm{x}$ und $\sigma_\mathrm{x}\sigma_\mathrm{z} = -i\sigma_\mathrm{y}$ also

$$\sigma_\mathrm{x}^{(1,1)}\sigma_\mathrm{y}^{(3,1)}\sigma_\mathrm{y}^{(4,1)}\sigma_\mathrm{z}^{(5,1)}\left|\phi\right\rangle \quad = \quad \kappa^{(1,1)}\kappa^{(3,1)}\kappa^{(4,1)}\left|\phi\right\rangle$$

Nun wendet man auf den Zustand den Messprojektor

$$\mathcal{P}_\mathrm{Hadamard} = \prod_{i\in\{2,3,4\}\times\{1\}} \frac{\mathbb{1}^{(i)} + \mu^{(i)}\sigma_\mathrm{y}^{(i)}}{2}$$

an, der den Zustand in den nach der Messung verwandelt:

$$\begin{aligned}
\mathcal{P}_\mathrm{Hadamard}\sigma_\mathrm{x}^{(1,1)}\sigma_\mathrm{y}^{(3,1)}\sigma_\mathrm{y}^{(4,1)}\sigma_\mathrm{z}^{(5,1)}\left|\phi\right\rangle &= \mathcal{P}_\mathrm{Hadamard}\kappa^{(1,1)}\kappa^{(3,1)}\kappa^{(4,1)}\left|\phi\right\rangle \\
\sigma_\mathrm{x}^{(\mathrm{a})}\sigma_\mathrm{z}^{(\mathrm{out(a)})}\prod_{i\in\{2,3,4\}\times\{1\}} \frac{\mathbb{1}^{(i)} + \mu^{(i)}\sigma_\mathrm{y}^{(i)}}{2}\sigma_\mathrm{y}^{(3,1)}\sigma_\mathrm{y}^{(4,1)}\left|\phi\right\rangle &= \kappa^{(1,1)}\kappa^{(3,1)}\kappa^{(4,1)}\mathcal{P}_\mathrm{Hadamard}\left|\phi\right\rangle
\end{aligned}$$

mit $\frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_\mathrm{y}^{(i)}}{2}\sigma_\mathrm{y}^{(i)} = \mu^{(i)}\frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_\mathrm{y}^{(i)}}{2}$

$$\begin{aligned}
\mu^{(3,1)}\mu^{(4,1)}\sigma_\mathrm{x}^{(\mathrm{a})}\sigma_\mathrm{z}^{(\mathrm{out(a)})}\prod_{i\in\{2,3,4\}\times\{1\}} \frac{\mathbb{1}^{(i)} + \mu^{(i)}\sigma_\mathrm{y}^{(i)}}{2}\left|\phi\right\rangle &= \kappa^{(1,1)}\kappa^{(3,1)}\kappa^{(4,1)}\mathcal{P}_\mathrm{Hadamard}\left|\phi\right\rangle \\
\sigma_\mathrm{x}^{(\mathrm{a})}\sigma_\mathrm{z}^{(\mathrm{out(a)})}\mathcal{P}_\mathrm{Hadamard}\left|\phi\right\rangle &= \mu^{(3,1)}\mu^{(4,1)}\kappa^{(1,1)}\kappa^{(3,1)}\kappa^{(4,1)}\mathcal{P}_\mathrm{Hadamard}\left|\phi\right\rangle
\end{aligned}$$

Mit $\left|\phi_\mathrm{Hadamard}\right\rangle = \mathcal{P}_\mathrm{Hadamard}\left|\phi\right\rangle$ und $\alpha_\mathrm{Hadamard} = \mu^{(3,1)}\mu^{(4,1)}\kappa^{(1,1)}\kappa^{(3,1)}\kappa^{(4,1)}$ also

$$\sigma_\mathrm{x}^{(\mathrm{a})}\sigma_\mathrm{z}^{(\mathrm{out(a)})}\left|\phi_\mathrm{Hadamard}\right\rangle \quad = \quad \alpha_\mathrm{Hadamard}\left|\phi_\mathrm{Hadamard}\right\rangle$$

Da $U_\mathrm{Hadamard}U_\mathrm{Hadamard}^\dagger = \mathbb{1}$ und $\sigma_\mathrm{z}U_\mathrm{Hadamard} = U_\mathrm{Hadamard}\sigma_\mathrm{x}$ gilt

$$\begin{aligned}
\sigma_\mathrm{x}^{(\mathrm{a})}\sigma_\mathrm{z}^{(\mathrm{out(a)})}U_\mathrm{Hadamard}^{(\mathrm{out(a)})}U_\mathrm{Hadamard}^{(\mathrm{out(a)})\dagger}\left|\phi_\mathrm{Hadamard}\right\rangle &= \alpha_\mathrm{Hadamard}\left|\phi_\mathrm{Hadamard}\right\rangle \\
\sigma_\mathrm{x}^{(\mathrm{a})}U_\mathrm{Hadamard}^{(\mathrm{out(a)})}\sigma_\mathrm{x}^{(\mathrm{out(a)})}U_\mathrm{Hadamard}^{(\mathrm{out(a)})\dagger}\left|\phi_\mathrm{Hadamard}\right\rangle &= \alpha_\mathrm{Hadamard}\left|\phi_\mathrm{Hadamard}\right\rangle
\end{aligned}$$

Nun zur σ_z-Vorraussetzung:

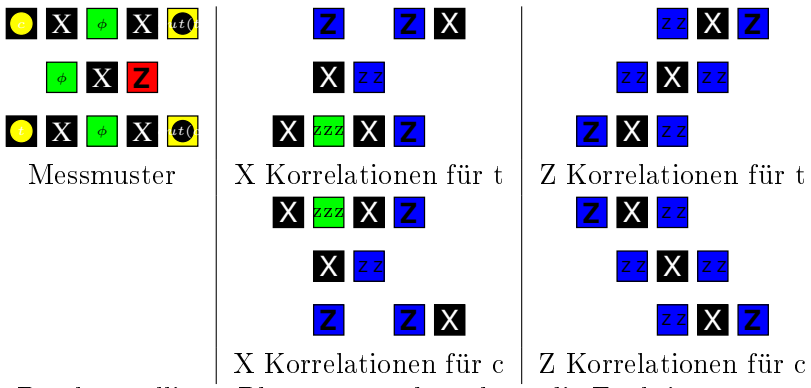
$$\begin{aligned}
\sigma_z^{(1,1)}\sigma_x^{(2,1)}\sigma_z^{(3,1)}\sigma_z^{(2,1)}\sigma_x^{(3,1)}\sigma_x^{(5,1)}\left|\phi\right\rangle &= \kappa^{(1,1)}\kappa^{(2,1)}\kappa^{(3,1)}\kappa^{(5,1)}\left|\phi\right\rangle \\
\sigma_z^{(1,1)}\sigma_y^{(2,1)}\sigma_y^{(3,1)}\sigma_x^{(5,1)}\left|\phi\right\rangle &= \kappa^{(1,1)}\kappa^{(2,1)}\kappa^{(3,1)}\kappa^{(5,1)}\left|\phi\right\rangle \\
\mathcal{P}_{\text{Hadamard}}\sigma_z^{(1,1)}\sigma_y^{(2,1)}\sigma_y^{(3,1)}\sigma_x^{(5,1)}\left|\phi\right\rangle &= \mathcal{P}_{\text{Hadamard}}\kappa^{(1,1)}\kappa^{(2,1)}\kappa^{(3,1)}\kappa^{(5,1)}\left|\phi\right\rangle \\
\prod_{i\in\{2,3,4\}\times\{1\}}\frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_y^{(i)}}{2}\sigma_z^{(1,1)}\sigma_y^{(2,1)}\sigma_y^{(3,1)}\sigma_x^{(5,1)}\left|\phi\right\rangle &= \kappa^{(1,1)}\kappa^{(2,1)}\kappa^{(3,1)}\kappa^{(5,1)}\mathcal{P}_{\text{Hadamard}}\left|\phi\right\rangle \\
\sigma_z^{(1,1)}\sigma_x^{(5,1)}\prod_{i\in\{2,3,4\}\times\{1\}}\frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_y^{(i)}}{2}\sigma_y^{(2,1)}\sigma_y^{(3,1)}\left|\phi\right\rangle &= \kappa^{(1,1)}\kappa^{(2,1)}\kappa^{(3,1)}\kappa^{(5,1)}\mathcal{P}_{\text{Hadamard}}\left|\phi\right\rangle \\
\sigma_z^{(1,1)}\sigma_x^{(5,1)}\mu^{(2,1)}\mu^{(3,1)}\prod_{i\in\{2,3,4\}\times\{1\}}\frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_y^{(i)}}{2}\left|\phi\right\rangle &= \kappa^{(1,1)}\kappa^{(2,1)}\kappa^{(3,1)}\kappa^{(5,1)}\mathcal{P}_{\text{Hadamard}}\left|\phi\right\rangle \\
\sigma_z^{(1,1)}\sigma_x^{(5,1)}\mu^{(2,1)}\mu^{(3,1)}\mathcal{P}_{\text{Hadamard}}\left|\phi\right\rangle &= \kappa^{(1,1)}\kappa^{(2,1)}\kappa^{(3,1)}\kappa^{(5,1)}\mathcal{P}_{\text{Hadamard}}\left|\phi\right\rangle \\
\sigma_z^{(a)}\sigma_x^{(out(a))}\mathcal{P}_{\text{Hadamard}}\left|\phi\right\rangle &= \mu^{(2,1)}\mu^{(3,1)}\kappa^{(1,1)}\kappa^{(2,1)}\kappa^{(3,1)}\kappa^{(5,1)}\mathcal{P}_{\text{Hadamard}}\left|\phi\right\rangle
\end{aligned}$$

Mit $\beta = \mu^{(2,1)}\mu^{(3,1)}\kappa^{(1,1)}\kappa^{(2,1)}\kappa^{(3,1)}\kappa^{(5,1)}$, $\left|\phi_{\text{Hadamard}}\right\rangle = \mathcal{P}_{\text{Hadamard}}\left|\phi\right\rangle$ und $\sigma_x\mathcal{U}_{\text{Hadamard}} = \mathcal{U}_{\text{Hadamard}}\sigma_z$ erhält man das Ergebnis

$$\begin{aligned}
\sigma_z^{(a)}\sigma_x^{(out(a))}\mathcal{P}_{\text{Hadamard}}\left|\phi\right\rangle &= \mu^{(2,1)}\mu^{(3,1)}\kappa^{(1,1)}\kappa^{(2,1)}\kappa^{(3,1)}\kappa^{(5,1)}\mathcal{P}_{\text{Hadamard}}\left|\phi\right\rangle \\
\sigma_z^{(a)}\sigma_x^{(out(a))}\left|\phi_{\text{Hadamard}}\right\rangle &= \beta\left|\phi_{\text{Hadamard}}\right\rangle \\
\sigma_z^{(a)}\sigma_x^{(out(a))}U_{\text{Hadamard}}^{(out(a))}U_{\text{Hadamard}}^{(out(a))\dagger}\left|\phi_{\text{Hadamard}}\right\rangle &= \beta\left|\phi_{\text{Hadamard}}\right\rangle \\
\sigma_z^{(a)}U_{\text{Hadamard}}^{(out(a))}\sigma_z^{(out(a))}U_{\text{Hadamard}}^{(out(a))\dagger}\left|\phi_{\text{Hadamard}}\right\rangle &= \beta\left|\phi_{\text{Hadamard}}\right\rangle
\end{aligned}$$

Damit sind, weil $\text{in} = \{a\} = \{(1,1)\}$ und $\text{out}((1,1)) = (5,1)$, die Vorrausetzungen des Beweises 4.1 erfüllt und der Quantenzustand führt eine Hadamard-Transformation auf dem Qubit a aus, wenn man es in X misst. $\tau_x^a = \alpha$ und $\tau_z^a = \beta$.

4.2.7 Kontrolliertes Phasengatter mit Swap (CPG)



Das kontrollierte Phasengatter berechnet die Funktion:

$$\left|0^{(c)}\right\rangle\left\langle 0^{(c)}\right|\otimes\mathbb{1}^{(t)} + \left|1^{(c)}\right\rangle\left\langle 1^{(c)}\right|\otimes\left(\left|0^{(t)}\right\rangle\left\langle 0^{(t)}\right| + e^{i\varphi}\left|1^{(t)}\right\rangle\left\langle 1^{(t)}\right|\right)$$

Die Vertauschung von c und t auf den Ausgängen wird - wie schon beim Swap-Gatter - durch die Definiton von out repräsentiert und taucht deshalb in der Formel nicht auf.

Aus diesem Gatter [2] wird die Quantenfouriertransformation aufgebaut. Es hat schiefe Messwinkel und erhöht deshalb die Zeitkomplexität, da man zuerst die Beiprodukte für die Eingabe kennen muss um die korrekten Winkel zu bestimmen.

Eine andere Möglichkeit, die Funktion darzustellen, ist

$$e^{i\varphi}\left|1^{(c)}1^{(t)}\right\rangle\left\langle 1^{(c)}1^{(t)}\right| + \sum_{(i,j)\in\{0,1\}^2-(1,1)}\left|i^{(c)}j^{(t)}\right\rangle\left\langle i^{(c)}j^{(t)}\right|$$

Hier ist die vollständige Symmetrie zwischen den beiden Eingängen sofort zu erkennen.

Für den Beweis wird die Transformation U_{CPG} noch in einer anderen Form geschrieben

$$\begin{aligned} U_{CPG}^{(in)} &\cong e^{i\varphi\frac{1^{(c)}-\sigma_z^{(c)}}{2}\frac{1^{(t)}-\sigma_z^{(t)}}{2}} \\ &= e^{i\frac{\varphi}{4}(1^{(c,t)}-\sigma_z^{(c)}-\sigma_z^{(t)}-\sigma_z^{(c)}\sigma_z^{(t)})} \\ &\cong e^{-i\frac{\varphi}{4}\sigma_z^{(c)}}e^{-i\frac{\varphi}{4}\sigma_z^{(t)}}e^{-i\frac{\varphi}{4}\sigma_z^{(c)}\sigma_z^{(t)}} \end{aligned}$$

die sich leicht als äquivalent mit den obigen erkennen lässt. In der letzten Zeile wurde die globale Phase weggelassen.

Es gibt eine gemeinsame Korrelationsrelation zum Beleg der X-Vorraussetzungen

$$\forall i \in \text{in}.\sigma_x^{(i)}U_{\text{CPG}(\varphi)}^{\text{out(in)}}\sigma_x^{\text{out}(i)}U_{\text{CPG}(\varphi)}^{\dagger\text{out(in)}}\left|\phi_{\text{CPG}(\varphi)}\right\rangle = \tau_x^{(i)}\left|\phi_{\text{CPG}(\varphi)}\right\rangle$$

Sie lautet

$$\sigma_z^{(2,2)}\ \sigma_x^{(3,2)}\sigma_x^{(4,1)}\sigma_x^{(4,3)}\sigma_z^{(5,1)}\sigma_z^{(5,3)}\left|\phi\right\rangle = \kappa^{(3,2)}\kappa^{(4,1)}\kappa^{(4,3)}\left|\phi\right\rangle$$

Definiert man $\kappa = \kappa^{(3,2)}\kappa^{(4,1)}\kappa^{(4,3)}$ folgt

$$\begin{aligned} \sigma_x^{(3,2)}\sigma_x^{(4,1)}\sigma_x^{(4,3)}\sigma_z^{(5,1)}\sigma_z^{(5,3)}\left|\phi\right\rangle &= \sigma_z^{(2,2)}\kappa\left|\phi\right\rangle \\ (\kappa\sigma_z^{(2,2)} - \sigma_x^{(3,2)}\sigma_x^{(4,1)}\sigma_x^{(4,3)}\sigma_z^{(5,1)}\sigma_z^{(5,3)})\left|\phi\right\rangle &= 0 \\ \forall\alpha.e^{i\alpha(\kappa\sigma_z^{(2,2)}-\sigma_x^{(3,2)}\sigma_x^{(4,1)}\sigma_x^{(4,3)}\sigma_z^{(5,1)}\sigma_z^{(5,3)})}\left|\phi\right\rangle &= \left|\phi\right\rangle \end{aligned}$$

Da die Operatoren kommutieren, gilt

$$\forall\alpha.e^{i\alpha\kappa\sigma_z^{(2,2)}}e^{-i\alpha\sigma_x^{(3,2)}\sigma_x^{(4,1)}\sigma_x^{(4,3)}\sigma_z^{(5,1)}\sigma_z^{(5,3)}}\left|\phi\right\rangle = \left|\phi\right\rangle$$

Einsetzen in die X-Korrelation für den Pin t

[2] von Dan Browne

$$
\begin{aligned}
\sigma_{\mathrm{x}}^{(1,1)}\sigma_{\mathrm{x}}^{(5,3)}\sigma_{\mathrm{x}}^{(2,2)}\sigma_{\mathrm{x}}^{(3,3)}\left|\phi\right\rangle &= \kappa^{(1,1)}\kappa^{(5,3)}\kappa^{(2,2)}\kappa^{(3,3)}\left|\phi\right\rangle \\
\forall\alpha.e^{i\alpha\kappa\sigma_{\mathrm{z}}^{(2,2)}}e^{-i\alpha\sigma_{\mathrm{x}}^{(3,2)}\sigma_{\mathrm{x}}^{(4,1)}\sigma_{\mathrm{x}}^{(4,3)}\sigma_{\mathrm{z}}^{(5,1)}\sigma_{\mathrm{z}}^{(5,3)}}\sigma_{\mathrm{x}}^{(1,1)}\sigma_{\mathrm{x}}^{(5,3)}\sigma_{\mathrm{x}}^{(2,2)}\sigma_{\mathrm{x}}^{(3,3)}\left|\phi\right\rangle &= \kappa^{(1,1)}\kappa^{(5,3)}\kappa^{(2,2)}\kappa^{(3,3)}\left|\phi\right\rangle \\
\forall\alpha.e^{i\alpha\kappa\sigma_{\mathrm{z}}^{(2,2)}}e^{-i\alpha\sigma_{\mathrm{x}}^{(3,2)}\sigma_{\mathrm{x}}^{(4,1)}\sigma_{\mathrm{x}}^{(4,3)}\sigma_{\mathrm{z}}^{(5,1)}\sigma_{\mathrm{z}}^{(5,3)}}\sigma_{\mathrm{x}}^{(1,1)}\sigma_{\mathrm{x}}^{(5,3)}\sigma_{\mathrm{x}}^{(2,2)}\sigma_{\mathrm{x}}^{(3,3)} & \\
e^{i\alpha\kappa\sigma_{\mathrm{z}}^{(2,2)}}e^{-i\alpha\sigma_{\mathrm{x}}^{(3,2)}\sigma_{\mathrm{x}}^{(4,1)}\sigma_{\mathrm{x}}^{(4,3)}\sigma_{\mathrm{z}}^{(5,1)}\sigma_{\mathrm{z}}^{(5,3)}}\left|\phi\right\rangle &= \kappa^{(1,1)}\kappa^{(5,3)}\kappa^{(2,2)}\kappa^{(3,3)}\left|\phi\right\rangle \\
\forall\alpha.\sigma_{\mathrm{x}}^{(1,1)}e^{i\alpha\kappa\sigma_{\mathrm{z}}^{(2,2)}}e^{-i\alpha\sigma_{\mathrm{x}}^{(3,2)}\sigma_{\mathrm{x}}^{(4,1)}\sigma_{\mathrm{x}}^{(4,3)}\sigma_{\mathrm{z}}^{(5,1)}\sigma_{\mathrm{z}}^{(5,3)}}\sigma_{\mathrm{x}}^{(5,3)} & \\
e^{-i\alpha\sigma_{\mathrm{x}}^{(3,2)}\sigma_{\mathrm{x}}^{(4,1)}\sigma_{\mathrm{x}}^{(4,3)}\sigma_{\mathrm{z}}^{(5,1)}\sigma_{\mathrm{z}}^{(5,3)}}e^{i\alpha\kappa\sigma_{\mathrm{z}}^{(2,2)}}\sigma_{\mathrm{x}}^{(2,2)}\sigma_{\mathrm{x}}^{(3,3)} &= \kappa^{(1,1)}\kappa^{(5,3)}\kappa^{(2,2)}\kappa^{(3,3)}\left|\phi\right\rangle
\end{aligned}
$$

Aus der Korrelationsrelation des Clusters für Qubit (4,1) folgt

$$
\sigma_{\mathrm{z}}^{(3,1)}\sigma_{\mathrm{x}}^{(4,1)}\sigma_{\mathrm{z}}^{(4,2)}\sigma_{\mathrm{z}}^{(5,1)}\left|\phi\right\rangle = \left|\phi\right\rangle
$$

Nun wendet man auf den Zustand den Messprojektor

$$
\mathcal{P}_{\mathrm{CPG}(\varphi)} = \prod_{i\in\{2,3,4\}\times\{1,2,3\}} \mathcal{P}_{\mathrm{CPG}(\varphi)}|(i)
$$

mit

$$
\begin{aligned}
\mathcal{P}_{\mathrm{CPG}(\varphi)}|(i) &= \frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2}\text{für } i\in\{(2,1),(3,2),(4,3),(2,3),(4,1)\} \\
\mathcal{P}_{\mathrm{CPG}(\varphi)}|(4,2) &= \frac{\mathbb{1}^{(4,2)}+\mu^{(4,2)}\sigma_{\mathrm{z}}^{(4,2)}}{2} \\
\mathcal{P}_{\mathrm{CPG}(\varphi)}|(i) &= \frac{\mathbb{1}^{(i)}+\mu^{(i)}e^{i\varphi\sigma_{\mathrm{z}}^{(i)}}\sigma_{\mathrm{x}}^{(i)}e^{-i\varphi\sigma_{\mathrm{z}}^{(i)}}}{2}\text{für } i\in\{(3,1),(2,2),(3,3)\}
\end{aligned}
$$

an, der den Zustand $\left|\phi\right\rangle$ in $\left|\phi_{\mathcal{P}_{\mathrm{CPG}(\varphi)}}\right\rangle$, den nach der Messung, verwandelt:

$$
\begin{aligned}
\mathcal{P}_{\mathrm{CPG}(\varphi)}\sigma_{\mathrm{z}}^{(3,1)}\sigma_{\mathrm{x}}^{(4,1)}\sigma_{\mathrm{z}}^{(4,2)}\sigma_{\mathrm{z}}^{(5,1)}\left|\phi\right\rangle &= \mathcal{P}_{\mathrm{CPG}(\varphi)}\left|\phi\right\rangle \\
\sigma_{\mathrm{z}}^{(3,1)}\mu^{(4,1)}\mu^{(4,2)}\sigma_{\mathrm{z}}^{(5,1)}\left|\phi_{\mathcal{P}_{\mathrm{CPG}(\varphi)}}\right\rangle &= \left|\phi_{\mathcal{P}_{\mathrm{CPG}(\varphi)}}\right\rangle
\end{aligned}
$$

Damit

$$
\begin{aligned}
(-\mu^{(4,1)}\mu^{(4,2)}\sigma_{\mathrm{z}}^{(5,1)}+\sigma_{\mathrm{z}}^{(3,1)})\left|\phi_{\mathcal{P}_{\mathrm{CPG}(\varphi)}}\right\rangle &= 0 \\
e^{-i\alpha(\mu^{(4,1)}\mu^{(4,2)}\sigma_{\mathrm{z}}^{(5,1)}-\sigma_{\mathrm{z}}^{(3,1)})}\left|\phi_{\mathcal{P}_{\mathrm{CPG}(\varphi)}}\right\rangle &= \left|\phi_{\mathcal{P}_{\mathrm{CPG}(\varphi)}}\right\rangle \\
e^{-i\alpha\mu^{(4,1)}\mu^{(4,2)}\sigma_{\mathrm{z}}^{(5,1)}}e^{i\alpha\sigma_{\mathrm{z}}^{(3,1)}}\left|\phi_{\mathcal{P}_{\mathrm{CPG}(\varphi)}}\right\rangle &= \left|\phi_{\mathcal{P}_{\mathrm{CPG}(\varphi)}}\right\rangle
\end{aligned}
$$

Wendet man nun den Projektor auf die Eigenwertgleichung für das Pin t an und setzt obigen Term ein, so erhält man

$$\begin{aligned}
\mathcal{P}_{\mathrm{CPG}(\varphi)}\sigma_{\mathrm{x}}^{(1,1)}e^{i\alpha\kappa\sigma_{\mathrm{z}}^{(2,2)}}e^{-i\alpha\sigma_{\mathrm{x}}^{(3,2)}\sigma_{\mathrm{x}}^{(4,1)}\sigma_{\mathrm{x}}^{(4,3)}\sigma_{\mathrm{z}}^{(5,1)}\sigma_{\mathrm{z}}^{(5,3)}}\sigma_{\mathrm{x}}^{(5,3)} & \\
e^{-i\alpha\sigma_{\mathrm{x}}^{(3,2)}\sigma_{\mathrm{x}}^{(4,1)}\sigma_{\mathrm{x}}^{(4,3)}\sigma_{\mathrm{z}}^{(5,1)}\sigma_{\mathrm{z}}^{(5,3)}}e^{i\alpha\kappa\sigma_{\mathrm{z}}^{(2,2)}}\sigma_{\mathrm{x}}^{(2,2)}\sigma_{\mathrm{x}}^{(3,3)}\left|\phi\right\rangle &= \mathcal{P}_{\mathrm{CPG}(\varphi)}\kappa^{(1,1)}\kappa^{(5,3)}\kappa^{(2,2)}\kappa^{(3,3)}\left|\phi\right\rangle \\
\sigma_{\mathrm{x}}^{(1,1)}e^{i\alpha\kappa\sigma_{\mathrm{z}}^{(2,2)}}e^{-i\alpha\mu^{(3,2)}\mu^{(4,1)}\mu^{(4,3)}\sigma_{\mathrm{z}}^{(5,1)}\sigma_{\mathrm{z}}^{(5,3)}} & \\
\sigma_{\mathrm{x}}^{(5,3)}e^{i\alpha\mu^{(3,2)}\mu^{(4,1)}\mu^{(4,3)}\sigma_{\mathrm{z}}^{(5,1)}\sigma_{\mathrm{z}}^{(5,3)}}e^{i\alpha\kappa\sigma_{\mathrm{z}}^{(2,2)}} & \\
\mathcal{P}_{\mathrm{CPG}(\varphi)}|(3,1)e^{-i\alpha\mu^{(4,3)}\sigma_{\mathrm{z}}^{(3,1)}}\sigma_{\mathrm{x}}^{(3,1)}e^{i\alpha\mu^{(4,3)}\sigma_{\mathrm{z}}^{(3,1)}} & \\
\mathcal{P}_{\mathrm{CPG}(\varphi)}|(2,2)e^{-i\alpha\sigma_{\mathrm{z}}^{(2,2)}}\sigma_{\mathrm{x}}^{(2,2)}e^{i\alpha\sigma_{\mathrm{z}}^{(2,2)}} & \\
\mathcal{P}_{\mathrm{CPG}(\varphi)}|(\{2,3,4\}\times\{1,2,3\})-\{(3,1),(2,2)\}\left|\phi\right\rangle &= \kappa^{(1,1)}\kappa^{(5,3)}\kappa^{(2,2)}\kappa^{(3,3)}\left|\phi_{\mathcal{P}_{\mathrm{CPG}(\varphi)}}\right\rangle \\
\sigma_{\mathrm{x}}^{(1,1)}e^{i\varphi\kappa\mu^{(2,2)}\sigma_{\mathrm{z}}^{(5,1)}}e^{-i\varphi\mu^{(3,2)}\mu^{(4,1)}\mu^{(4,3)}\sigma_{\mathrm{z}}^{(5,1)}\sigma_{\mathrm{z}}^{(5,3)}} & \\
\sigma_{\mathrm{x}}^{(5,3)}e^{i\varphi\mu^{(3,2)}\mu^{(4,1)}\mu^{(4,3)}\sigma_{\mathrm{z}}^{(5,1)}\sigma_{\mathrm{z}}^{(5,3)}}e^{i\varphi\kappa\mu^{(2,2)}\sigma_{\mathrm{z}}^{(5,1)}}\mu^{(2,2)}\mu^{(3,2)}\left|\phi_{\mathcal{P}_{\mathrm{CPG}(\varphi)}}\right\rangle &= \kappa^{(1,1)}\kappa^{(5,3)}\kappa^{(2,2)}\kappa^{(3,3)}\left|\phi_{\mathcal{P}_{\mathrm{CPG}(\varphi)}}\right\rangle
\end{aligned}$$

Also

$$\sigma_{\mathrm{x}}^{(1,1)}U_{\mathrm{CPG}(\varphi)}\sigma_{\mathrm{x}}^{(5,3)}U_{\mathrm{CPG}(\varphi)}^{\dagger}=\mu^{(2,2)}\mu^{(3,2)}\kappa^{(1,1)}\kappa^{(5,3)}\kappa^{(2,2)}\kappa^{(3,3)}\left|\phi_{\mathcal{P}_{\mathrm{CPG}(\varphi)}}\right\rangle$$

Für das zweite Inputpin verläuft der Beweis analog.

Der Beweis für die Z Korrelationen ist einfacher und funktioniert nach dem gleichen Schema wie bei allen "geraden" Gattern.

Die Korrelationsrelation für t lautet

$$\sigma_{\mathrm{z}}^{(1,3)}\sigma_{\mathrm{x}}^{(2,3)}\ \sigma_{\mathrm{x}}^{(3,2)}\sigma_{\mathrm{x}}^{(4,1)}\sigma_{\mathrm{z}}^{(5,1)}\left|\phi\right\rangle = \kappa^{(2,3)}\kappa^{(3,2)}\kappa^{(4,1)}\left|\phi\right\rangle$$

mit $\mathrm{Korr}_{\mathrm{Z}}=\{(1,3),(2,3),(3,2),(4,1),(5,1)\}$, $\mathrm{Meas}=\{2,3,4\}\times\{1,2,3\}$, $\mathrm{Pins}=\{(1,3),(5,1)\}$, $\underline{\mathrm{Korr}_{\mathrm{Z}}}=\mathrm{Korr}_{\mathrm{Z}}\cap\mathrm{Meas}$ (also die auf die gemessenen Koordinaten beschränkte Menge) und $\overline{\mathrm{Korr}_{\mathrm{Z}}}=\mathrm{Meas}-\underline{\mathrm{Korr}_{\mathrm{Z}}}$ also

$$\sigma_{\mathrm{z}}^{(1,3)}\sigma_{\mathrm{z}}^{(5,1)}\prod_{i\in\underline{\mathrm{Korr}_{\mathrm{Z}}}}\sigma_{\mathrm{x}}^{(i)}\ \left|\phi\right\rangle = \prod_{i\in\underline{\mathrm{Korr}_{\mathrm{Z}}}}\kappa^{(i)}\left|\phi\right\rangle$$

Nun wendet man auf den Zustand den Messprojektor $\mathcal{P}_{\mathrm{CPG}(\varphi)}$ an, der den Zustand in den nach der Messung verwandelt:

$$\mathcal{P}_{\mathrm{CPG}(\varphi)}\sigma_{\mathrm{z}}^{(1,3)}\sigma_{\mathrm{z}}^{(5,1)}\prod_{i\in\underline{\mathrm{Korr_Z}}}\sigma_{\mathrm{x}}^{(i)}\,|\phi\rangle \;=\; \mathcal{P}_{\mathrm{CPG}(\varphi)}\prod_{i\in\underline{\mathrm{Korr_Z}}}\kappa^{(i)}\,|\phi\rangle$$

$$\sigma_{\mathrm{z}}^{(1,3)}\sigma_{\mathrm{z}}^{(5,1)}\prod_{i\in\underline{\mathrm{Korr_Z}}}\mathcal{P}_{\mathrm{CPG}(\varphi)}|(i)\sigma_{\mathrm{x}}^{(i)}\prod_{i\in\overline{\mathrm{Korr_Z}}}\mathcal{P}_{\mathrm{CPG}(\varphi)}|(i)\,|\phi\rangle \;=\; \prod_{i\in\underline{\mathrm{Korr_Z}}}\kappa^{(i)}\mathcal{P}_{\mathrm{CPG}(\varphi)}\,|\phi\rangle$$

$$\sigma_{\mathrm{z}}^{(1,3)}\sigma_{\mathrm{z}}^{(5,1)}\prod_{i\in\underline{\mathrm{Korr_Z}}}\frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2}\sigma_{\mathrm{x}}^{(i)}\prod_{i\in\overline{\mathrm{Korr_Z}}}\mathcal{P}_{\mathrm{CPG}(\varphi)}|(i)\,|\phi\rangle \;=\; \prod_{i\in\underline{\mathrm{Korr_Z}}}\kappa^{(i)}\mathcal{P}_{\mathrm{CPG}(\varphi)}\,|\phi\rangle$$

$$\sigma_{\mathrm{z}}^{(1,3)}\sigma_{\mathrm{z}}^{(5,1)}\prod_{i\,\in\,\underline{\mathrm{Korr_Z}}}\mu^{(i)}\frac{\mathbb{1}^{(i)}+\mu^{(i)}\sigma_{\mathrm{x}}^{(i)}}{2}\prod_{i\in\overline{\mathrm{Korr_Z}}}\mathcal{P}_{\mathrm{CPG}(\varphi)}|(i)\,|\phi\rangle \;=\; \prod_{i\in\underline{\mathrm{Korr_Z}}}\kappa^{(i)}\mathcal{P}_{\mathrm{CPG}(\varphi)}\,|\phi\rangle$$

$$\sigma_{\mathrm{z}}^{(1,1)}\sigma_{\mathrm{z}}^{(5,1)}\prod_{i\,\in\,\underline{\mathrm{Korr_Z}}}\mu^{(i)}$$

$$\prod_{i\,\in\,\underline{\mathrm{Korr_Z}}}\mathcal{P}_{\mathrm{CPG}(\varphi)}|(i)\prod_{i\in\overline{\mathrm{Korr_Z}}}\mathcal{P}_{\mathrm{CPG}(\varphi)}|(i)\,|\phi\rangle \;=\; \prod_{i\in\underline{\mathrm{Korr_Z}}}\kappa^{(i)}\mathcal{P}_{\mathrm{CPG}(\varphi)}\,|\phi\rangle$$

$$\sigma_{\mathrm{z}}^{(1,1)}\sigma_{\mathrm{z}}^{(5,1)}\prod_{i\,\in\,\underline{\mathrm{Korr_Z}}}\mu^{(i)}\mathcal{P}_{\mathrm{CPG}(\varphi)}\,|\phi\rangle \;=\; \prod_{i\in\underline{\mathrm{Korr_Z}}}\kappa^{(i)}\mathcal{P}_{\mathrm{CPG}(\varphi)}\,|\phi\rangle$$

$$\sigma_{\mathrm{z}}^{(\mathrm{t})}\sigma_{\mathrm{z}}^{(\mathrm{out(t)})}\mathcal{P}_{\mathrm{CPG}(\varphi)}\,|\phi\rangle = \prod_{i\,\in\,\underline{\mathrm{Korr_Z}}}\mu^{(i)}\prod_{i\in\underline{\mathrm{Korr_Z}}}\kappa^{(i)}\mathcal{P}_{\mathrm{CPG}(\varphi)}\,|\phi\rangle$$

mit $\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle := \mathcal{P}_{\mathrm{CPG}(\varphi)}\,|\phi\rangle$ dann

$$\sigma_{\mathrm{z}}^{(\mathrm{t})}\sigma_{\mathrm{z}}^{(\mathrm{out(t)})}\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle \;=\; \prod_{i\in\underline{\mathrm{Korr_Z}}}\mu^{(i)}\kappa^{(i)}\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle$$

Für Pin c wird analog aus

$$\sigma_{\mathrm{z}}^{(1,1)}\sigma_{\mathrm{x}}^{(2,1)}\;\sigma_{\mathrm{x}}^{(3,2)}\sigma_{\mathrm{x}}^{(4,3)}\sigma_{\mathrm{z}}^{(5,3)}\,|\phi\rangle \;=\; \kappa^{(2,1)}\kappa^{(3,2)}\kappa^{(4,3)}\,|\phi\rangle$$

die Gleichung

$$\sigma_{\mathrm{z}}^{(\mathrm{c})}\sigma_{\mathrm{z}}^{(\mathrm{out(c)})}\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle \;=\; \prod_{i\in\underline{\mathrm{Korr_Z}}}\mu^{(i)}\kappa^{(i)}\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle$$

mit $\mathrm{Korr_Z} = \{(1,1),(2,1),(3,2),(4,3),(5,3)\}$ abgeleitet.

Sei $U_{\mathrm{CPG}(\varphi)} = \left|0^{(c)}\right\rangle\left\langle 0^{(c)}\right| \otimes \mathbb{1}^{(t)} + \left|1^{(c)}\right\rangle\left\langle 1^{(c)}\right| \otimes (\left|0^{(t)}\right\rangle\left\langle 0^{(t)}\right| + e^{i\varphi}\left|1^{(t)}\right\rangle\left\langle 1^{(t)}\right|)$. Es gilt

$$
\begin{aligned}
& U_{\mathrm{CPG}(\varphi)}\sigma_{\mathrm{z}}^{(\mathrm{out(c)})} \\
= \; & (\left|0^{(c)}\right\rangle\left\langle 0^{(c)}\right| \otimes \mathbb{1}^{(t)} + \left|1^{(c)}\right\rangle\left\langle 1^{(c)}\right| \otimes (\left|0^{(t)}\right\rangle\left\langle 0^{(t)}\right| + e^{i\varphi}\left|1^{(t)}\right\rangle\left\langle 1^{(t)}\right|))\sigma_{\mathrm{z}}^{(\mathrm{out(c)})} \\
= \; & \left|0^{(c)}\right\rangle\left\langle 0^{(c)}\right| \otimes \mathbb{1}^{(t)} - \left|1^{(c)}\right\rangle\left\langle 1^{(c)}\right| \otimes (\left|0^{(t)}\right\rangle\left\langle 0^{(t)}\right| + e^{i\varphi}\left|1^{(t)}\right\rangle\left\langle 1^{(t)}\right|) \\
= \; & (\sigma_{\mathrm{z}}^{(\mathrm{out(c)})}\left|0^{(c)}\right\rangle)\left\langle 0^{(c)}\right| \otimes \mathbb{1}^{(t)} + (\sigma_{\mathrm{z}}^{(\mathrm{out(c)})}\left|1^{(c)}\right\rangle)\left\langle 1^{(c)}\right| \otimes (\left|0^{(t)}\right\rangle\left\langle 0^{(t)}\right| + e^{i\varphi}\left|1^{(t)}\right\rangle\left\langle 1^{(t)}\right|) \\
= \; & \sigma_{\mathrm{z}}^{(\mathrm{out(c)})}(\left|0^{(c)}\right\rangle\left\langle 0^{(c)}\right| \otimes \mathbb{1}^{(t)} + \left|1^{(c)}\right\rangle\left\langle 1^{(c)}\right| \otimes (\left|0^{(t)}\right\rangle\left\langle 0^{(t)}\right| + e^{i\varphi}\left|1^{(t)}\right\rangle\left\langle 1^{(t)}\right|)) \\
= \; & \sigma_{\mathrm{z}}^{(\mathrm{out(c)})}U_{\mathrm{CPG}(\varphi)}
\end{aligned}
$$

und somit

$$
\begin{aligned}
\sigma_{\mathrm{z}}^{(\mathrm{c})}\sigma_{\mathrm{z}}^{(\mathrm{out(c)})}\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle &= \prod_{i\in\underline{\mathrm{Korr}_{\mathrm{Z}}}} \mu^{(i)}\kappa^{(i)}\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle \\
\sigma_{\mathrm{z}}^{(\mathrm{c})}U_{\mathrm{CPG}(\varphi)}^{\mathrm{out(in)}}U_{\mathrm{CPG}(\varphi)}^{\dagger\mathrm{out(in)}}\sigma_{\mathrm{z}}^{(\mathrm{out(c)})}\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle &= \prod_{i\in\underline{\mathrm{Korr}_{\mathrm{Z}}}} \mu^{(i)}\kappa^{(i)}\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle \\
\sigma_{\mathrm{z}}^{(\mathrm{c})}U_{\mathrm{CPG}(\varphi)}^{\mathrm{out(in)}}\sigma_{\mathrm{z}}^{(\mathrm{out(c)})}U_{\mathrm{CPG}(\varphi)}^{\dagger\mathrm{out(in)}}\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle &= \prod_{i\in\underline{\mathrm{Korr}_{\mathrm{Z}}}} \mu^{(i)}\kappa^{(i)}\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle
\end{aligned}
$$

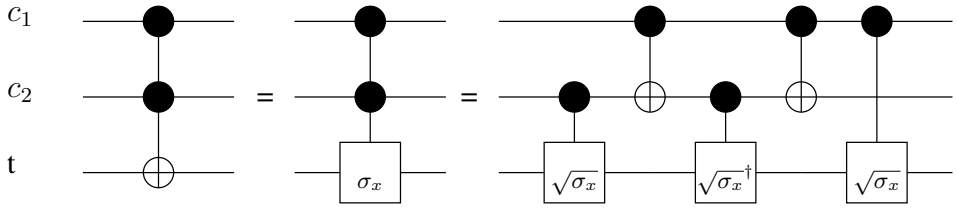
Dann gilt nach obigen Herleitungen und mit $\mathrm{in} = \{\mathrm{c},\mathrm{t}\} = \{(1,1),(1,3)\}$, $\mathrm{out}((1,3)) = (5,1)$ und $\mathrm{out}((1,1)) = (5,3)$ das gewünschte Ergebnis

$$\forall i \in \mathrm{in}.\sigma_{\mathrm{z}}^{(i)}U_{\mathrm{CPG}(\varphi)}^{\mathrm{out(in)}}\sigma_{\mathrm{z}}^{\mathrm{out}(i)}U_{\mathrm{CPG}(\varphi)}^{\dagger\mathrm{out(in)}}\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle = \tau_{\mathrm{z}}^{(i)}\left|\phi_{\mathrm{CPG}(\varphi)}\right\rangle$$

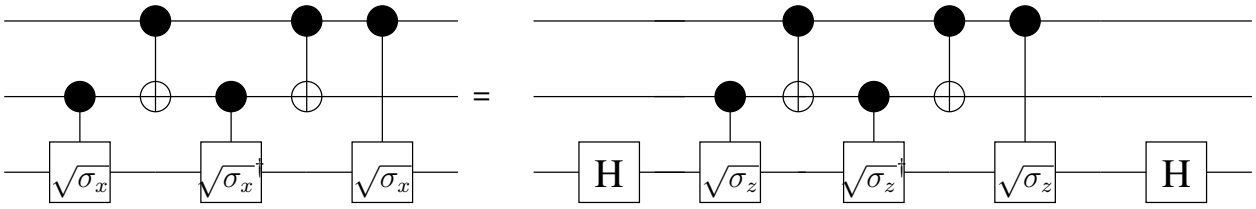
Damit sind die notwendigen Eigenschaften des Messmusters für das kontrollierte Phasengatter mit Swap gezeigt.

4.2.8 Toffoli, CCNOT-Gatter

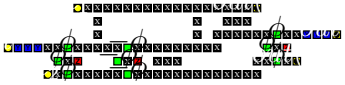
Dieses Gatter wird nach dem Schema des Artikels [1], S.14 aufgebaut.



Mit Hadamard-Transformationen, um aus σ_x-Rotationen σ_z-(Phasen-)-Rotationen zu machen und der Verwendung von kontrollierten Phasengattern mit Phase $\sqrt{\sigma_z} = e^{i\pi/2\sigma_z}$ entsteht folgendes Quantennetzwerk

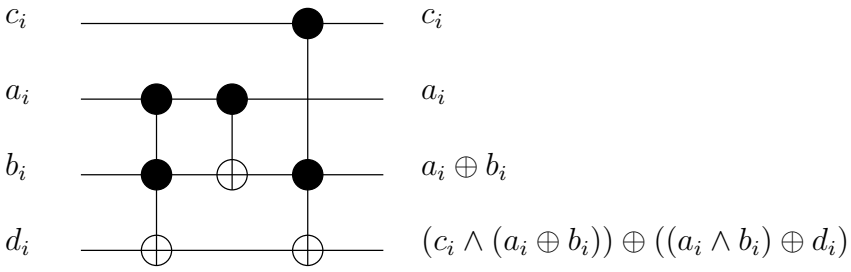


und daraus dieses Messmuster:



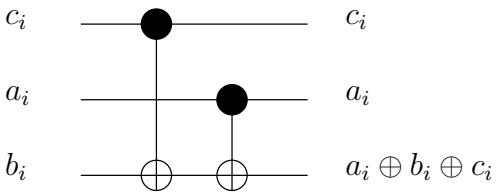
4.2.9 Carry-(Übertrag-)Gatter

Das Carry ist in der unitären Darstellung überraschend kompliziert. Es besteht aus einem Hilfsqubit (ancilla) $|0\rangle$ und zwei Toffoli-Gattern, die folgende Funktion berechnen: $(c \wedge (a \oplus b)) \oplus ((a \oplus b) \wedge d)$, wobei "d" das Hilfsqubit ist.



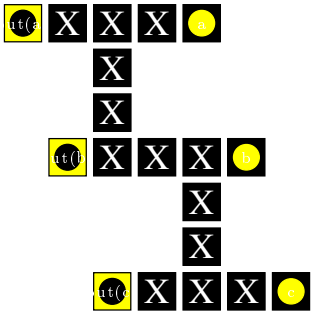
Und das dazugehörige Messmuster

4.2.10 Addierer

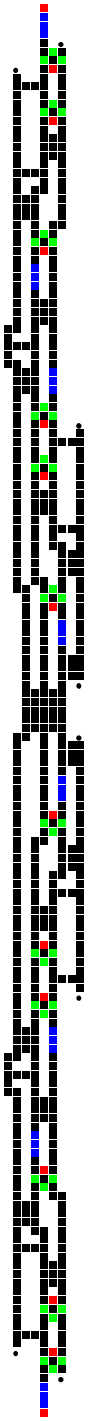


Der Halbaddierer, der nur das untere Bit des Ergebnisses, also keinen Übertrag berechnet, besteht nur aus zwei CNOTs, die die einbitige Summe $a \oplus b \oplus c$ berechnen, wie schon hier dargestellt 2.3.3.3.

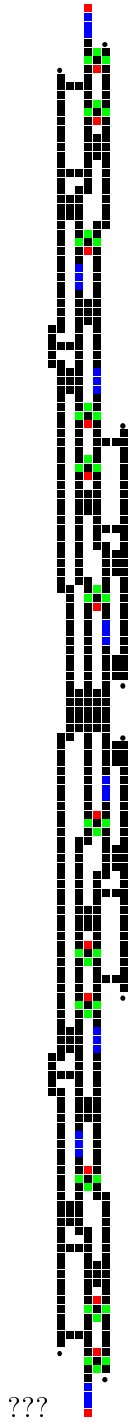
Das Messmuster



Der Volladdierer - wie sein konventionelles Gegenstück - besteht aus einem Carry und einem Halbaddierer. Damit das Gatter eine unitäre Transformation ausführt, muss man die Carryberechnung wieder umkehren um so das Hilfsqubit zu entschränken.



Ein 4-Bit-Addierer ist nur die Zusammenschaltung der Übertragsbits von vier Volladdierern.

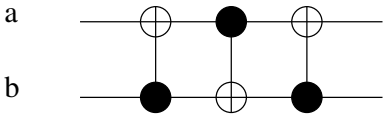


4.2.11 Kontrolliertes Swap-Gatter

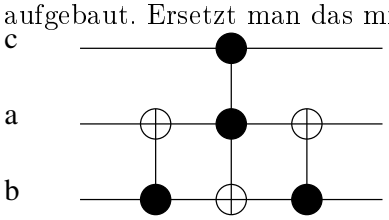
Zum Aufbau des Exponentiationsnetzwerkes brauche ich an einigen Stellen bedingt ausgeführte Operationen. Das wird durch ein kontrolliertes Swap-Gatter erreicht, das am

einen Eingang das Resultat der Operation, am anderen den unveränderten Zustand erhält und vertauscht oder nicht vertauscht, je nach Zustand der Kontrolle. An einem Ausgang schliesst man dann das weitere Netzwerk an, der andere ist Quantenmüll, der offen bleibt.

Das Gatter ist nach dem klassischen Vertauschungsnetzwerk

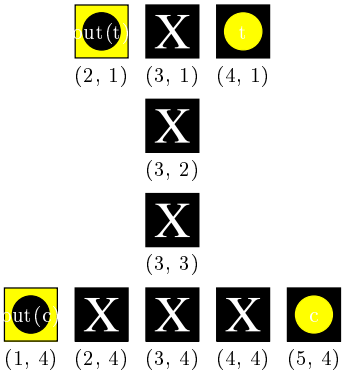


aufgebaut. Ersetzt man das mittlere CNOT durch ein Toffoli,



das man ja auch als bedingtes CNOT betrachten kann, so sieht man: Ist die obere (Kontroll-)Leitung auf $|0\rangle$, so ist das Toffoli auf den unteren zwei Leitungen die Identität - die CNOTs heben sich auf. Ist die Kontrollleitung auf $|1\rangle$, so wirkt das Gatter auf den unteren zwei Leitungen wie ein normales Vertauschungsgatter.

Das Messmuster ist einfach aus dem Toffoli und den CNOTs zusammengesetzt.



4.3 Quantennetzwerke als Messmuster

4.3.1 Quantenfouriertransformation

Dieses Netzwerk ist ein fehlerbehaftetes, aus kontrollierten Phasengattern und Hadamardtransformationen von einem Skript erstellte Variante einer Quantenfouriertransformation.

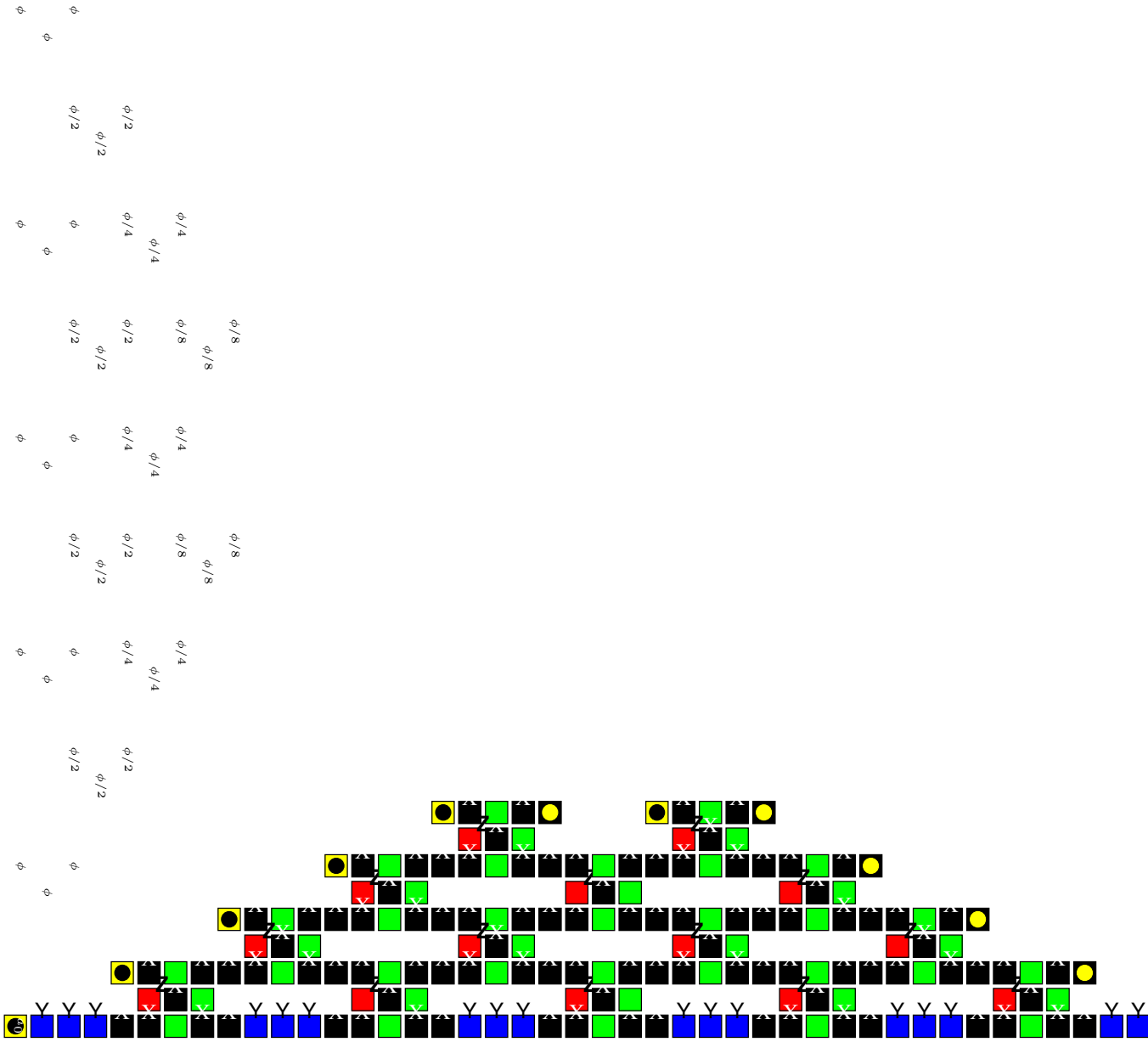


Abbildung 4.1: $errQFT_{2^7}^k$

5 Implementierung des Algorithmus von Shor

5.0.2 Zahlentheoretische Motivation

Der Faktorisierungsalgorithmus von Shor besteht aus zwei Teilen: Dem Bestimmen der Ordnung eines Elements der Gruppe $\mathbb{Z}/N$ und daraus dann die Berechnung eines Faktors von N. Der erste Teil lässt sich auf Quantencomputern mit Shors Algorithmus exponentiell schneller lösen als auf klassischen mit den bekannten Algorithmen und ist der eigentlich quanteninformatische während die Faktorisierung nur *eine* Anwendung dafür ist. Eine andere ist das Finden diskreter Logarithmen, oder generell das Aufspüren versteckter Gruppen.

5.0.2.1 Faktorisierung

Der Algorithmus für den Quantencomputer ermittelt die Periode k einer Funktion $f_{a,N}(x) = a^x \bmod N$. Ist a ein Element aus der multiplikativen Gruppe $\{x \in 1, ..., N | \mathrm{ggT}(x, N) = 1\}$, dann gilt: $a^k \bmod N = 1$, da $1 \equiv a^{-r}a^r \equiv a^{-r}f_{a,N}(r) \equiv a^{-r}f_{a,N}(k+r) \equiv a^{-r}a^{k+r} \equiv a^k \pmod N$, die Periode von f ist also gleich der Ordnung von a. Ist nun k gerade, dann gilt $(a^{k/2})^2 - 1 \equiv 0 \bmod N$. Damit ist $(a^{k/2}-1)(a^{k/2}+1) \equiv 0 \bmod N$ und $\mathrm{ggT}(a^{k/2}-1, N)$ ist ein Faktor von N. Bei ungeradem k wählt man sich ein neues a und beginnt von vorne (siehe [12]).

5.1 Exponentiation

Die Periode k der Exponentiationsfunktion $f_{a,N}(x) = a^x \bmod N$ hängt nicht von einem Vorfaktor v ab, da $f'_{a,N}(x) = va^x \bmod N$ andere Werte, aber die gleiche Periode wie f hat. Genausowenig ist k, wie von Ekert in [6] erläutert, abhängig von dem gemessenen zufälligen Funktionswert m, da für $m_1 = a^{r_1}$ und $m_2 = a^{r_2}$ das Exponentenregister in einer Überlagerung $lk+r_1$ bzw. $lk+r_2$ besteht, sich also in der Periode nicht unterscheiden. Das Messen des Funktionswertes von f ist also nicht notwendig.

5.2 Komplexitätsbetrachtung

Eine Faktorisierung mit dem hier angegebenen Netzwerk benötigt im wesentlichen n^3 Addierergatter. Das hier vorgestellte Gatter ist ca. 2000 Clusterqubits groß, d.h. für eine interessante Zerlegung einer Zahl mit 250 Bits benötigte man einen Cluster mit 30 Milliarden Qubits. Das lässt sich allerdings mit bekannten Mitteln noch stark verbessern:

Zum einen kann man an den Gattern und Zusammenschaltungen lokale Optimierungen durchführen, also überflüssiges Quantenkabel entfernen etc. Damit erreicht man eine Verbesserung vielleicht um den Faktor 2-5. Viel bedeutender ist daher der Einsatz eines besseren Multiplizieralgorithmus, z.B. der von Toom, der statt n^2 nur eine Komplexität von $n \log n$ hat. Damit würde ein Schaltkreis für eine 250-Bit-Zahl nur mehr 1/30 der "normalen" Größe betragen.

6 Schlusswort

Dieses Schlusswort ist vor allem eine Liste der Dinge, die ich für diese Arbeit angedacht habe, die aber aus Zeitgründen nicht darin auftauchen. Viele der hier vorgestellten Verbesserungen sind praktischer Natur, machen die Messmuster kleiner und die Entwicklung einfacher, sind aber für das theoretische Verständnis bedeutungslos. Sie zu implemtieren macht daher nur Sinn, wenn konkrete Realisierungen eventuell darauf zurückgreifen.

- Simulation: Die Schwierigkeit besteht in der viel größeren - im Vergleich zu Verfahren, die während der Berechnung auch ver- und nicht nur entschränken - Anzahl von Qubits, die selbst für einfachste Rechnungen gebraucht werden. Das macht wegen der doppelt-exponentiellen Komplexität so große Probleme, dass die "naiven" Simulationsverfahren nicht verwendet werden können, sondern durch Ausnutzen spezieller Beschränkungen des Modells zuerst die Größe auf ein handhabbares Maß gestutzt werden muss, um überhaupt etwas berechnen zu können.
- Verifikation von elementaren Gattern durch Nachrechnen von handgemachten Korrelationsrelationen oder durch brute-force erzeugte
- Verifikation von Netzwerken durch Komposition
- Verifikation von Netzwerken durch Vereinigung der Korrelationszentren
- direkte Verifikation von Netzwerken wie elementare Gatter
- globale Optimierung: Hier vor allem der Einsatz eines besseren Multiplikationsverfahrens wie z.B. das von Toom (Knuth, Bronstein?)
- Einführen einer Notation für Netzwerke wie die Beschreibungsdateien mit Messmustern
- für elementare Gatter
- Optimierung auf Netzwerkebene
- Optimierung der Messmuster (z.B. QFT)
- Fehlerabschätzung
- Berechnung der Propagationsrelationen und damit der Ergebnisse des Einwegquantencomputers
- Berechnung der Reihenfolge der Messungen

- Einbeziehen von Unregelmäßigkeiten im Cluster
- Müssen alle überflüssigen Ausgänge des Quantennetzwerkes 0 sein, oder lässt sich das herausrechnen
- Lösen vom Paradigma der Quantengatter, zumindest in der strikten Form, könnte andere Einsichten in die Mächtigkeit und Struktur der Verschränkungseigenschaften des Clusterzustandes bewirken. Z.B. die Betrachtung von zirkulären Mustern könnte leistungsfähigere Rechenschema ergeben. Auch die physikalische Basis wird eventuell durch ein Verzicht auf die nur aufgesetzten Konstruktionen eines Quantennetzwerks klarer.

7 Anhang

7.1 Programme

Hier die Listings der Programme, die ich für diese Arbeit geschrieben habe.

7.1.1 Der Composer zur Darstellung von Quantengattern für den Einwegquantencomputer

Geschrieben in python.

```
# Quantum Gate Composer for One-Way Quantum Computer

import figclasses, copy, operator, string, sys, os.path, pickle, os, stat

################ error definition ###################

class OverlapError(Exception):
  pass

class NoWayError(Exception):
  pass

class NoMountError(Exception):
  pass

def error(s):
  raise Exception(s)

def figdump(object, filename=None):
  if not filename: filename="/tmp/dump"+repr(object)+".fig"
  fd=open(filename, "w")
  figs=object.figobjects(["build", "place"], None, lambda c: str(c),
         figclasses.sigmas)

##  # calculating figure width to resize it appropriately and move them to (0,0)
##  ((xlu,ylu),(xrl,yrl))=figclasses.figssize(figs)
##  for o in figs: o.move(lambda (x,y),(xlu,ylu)=(xlu,ylu): (x-xlu, y-ylu))
##  mx=xrl-xlu  # calculating figure size to resize it appropriately
```

```
##  my=yrl-ylu
###    print "FIGSIZE for dump at ", o, " is ", ((xlu,ylu),(xrl,yrl)), \
###          " total: ", (mx,my)
##  factor=min(6000.0/mx,8400.0/my)
###  print "FACTOR figdump ", factor
##
##  # resizing figure
##  for o in figs:
##    o.move(lambda (x,y), factor=factor: (factor*x,factor*y))

  # writing it to file
  fd=open(filename, "w");
  figclasses.figfile.contents=figs
  fd.write(str(figclasses.figfile)+"\n")
  fd.close()
  print "created file "+filename+"."

######################## general functions ############################

def urbild(check, assarray): # returns all elements of assarray with
                             # check(value)==true
#  print "URBILD received VALUE ", value, " ARRAY ", assarray
  list=[]
  for (k,v) in assarray.items():
    if check(v): list.append((k,v))
#  print "URBILD returned ", list
  return list

def inverse(check, assarray): # returns single element of assarray with
                              # check(value)==true or raises an error
  o=urbild(check, assarray)
  if not len(o)==1:
    error("\""+str(check)+"\" filter returns "+\
          ((len(o) and "more than one.") or "none."))
  else: return o[0]

# recursive functions can only live on the top level, because the
# function doesn't find its own name if it is defined e.g. in another
# function
# so route goes here

# build recursively a quantum wire from x,y to x2,y2
def route((x,y), visited, level, (xt,yt), freepoint):
  if level>200: return None # speeds it up and makes recursion depth explicit
```

```
#  print level, "ROUTE ", (x,y), " TO ", (xt,yt)
  if (x,y)==(xt,yt): return []
  l=[(x-1,y),(x+1,y),(x,y-1),(x,y+1)]
  l.sort(lambda (xl,yl),(xr,yr),(xt,yt)=(xt,yt): \
           cmp((xl-xt)**2+(yl-yt)**2,(xr-xt)**2+(yr-yt)**2))
  for coord in [c for c in l if c not in visited]:
#    print level, "SELECTED ", coord, " FROM ", l
    if freepoint(coord, visited[:len(visited)-1]):
      v=visited
      v.append((x,y))
      result=route(coord, v, level+1, (xt,yt), freepoint)
#      print level, "RETURNING ", result
      if not result==None:
        r=[coord]
        r.extend(result)
        return r
#      print "!!! NEVER REACHED !!!"
  return None

# build recursively a gate, this allows backtracking of badly placed gates

global versions

def recursivemount(build, todolist, inandouttox=0, dopath=None,
                   donepath=None):
  if donepath==None: donepath=[]
  if todolist==[]:
#    print "recusivemount inmediatly returned ", build
    return build

  (gate, mounting, relabel)=todolist[0]
#  print "BUILD \n", build, "\n"
  transforms=build.matchlist(gate, mounting)
#  print "TRANSFORMS ", transforms

  # dopath is a list of path choices to reduce complexity
  if dopath: trange=[dopath[0]]; dopath=dopath[1:]
  else: trange=range(0,len(transforms))

  for i in trange:
    p=copy.copy(donepath); p.append(i)
    print len(todolist), " items to do ", i+1, "th transform of ",\
          len(transforms), " donepath: ", p
    current=copy.deepcopy(build)
```

```
    try: rubresult=current.rubbermount(gate, mounting, [transforms[i]],
                                       inandouttox)
    except NoMountError:
#      print "NoMountError!"
      continue # try next
    rubresult.relabel(relabel)

    if not rubresult.inoutreachable(): continue

    result=recursivemount(rubresult, todolist[1:], inandouttox,
                          dopath, p)
    if result:
      if result in map(lambda (p,r): r, versions):
        print repr(result), " already in versions.\n"
        # return None
      else:
        versions.append((p, result))
        print repr(result), " appended to versions.\n"
        # return None
    # else try next element of transforms
  else: # no element of transforms worked
#    print "NONE returned for complete fail to mount\n", gate,\
#          "\nvvvv to vvvv\n", current, "\n"
    return None

################ classes describing a measurement pattern #############

class Measurement:
  """representing a one atom measurement and possibly its graphical
     visualisation as a fig object"""
  def __init__(self, measurement, name="", figobject=None):
    self.measurement=measurement # string "X","Y", etc.
    self.figobject=figobject
    self.name=name

class Pattern:
  """a measurement pattern, usually read from a ASCII file, where X are
     X-measurements etc.
     Can be transformed (rotated, flipped, translated) and drawn as a fig
     file"""
  def __init__(self,list=[], symbolmapping={}):
    self.pattern={};
    self.size=(0,0);
    rc=0;
```

```
  for l in list:
    cc=0;
    for c in l:
      if not c==None:
        self.pattern[(cc,rc)]=Measurement(symbolmapping[c],c);
      cc=cc+1;
    rc=rc+1;
  self.adjustsize();

def __str__(self):
  """returns a string representation of the Pattern object"""

  keys=self.pattern.keys();
  keys.sort(lambda (x1,y1),(x2,y2): 2*cmp(y1,y2)+cmp(x1,x2));
  try: left=min(map(lambda (x,y): x, keys));
  except ValueError: left=0
  try: top=min(map(lambda (x,y): y, keys));
  except ValueError: top=0
  cc=left; rc=top; result="";
  for (a,b) in keys:
    while b>rc: cc=left; rc=rc+1; result=result+"\n";
    while a>cc: cc=cc+1; result=result+" ";
    towrite=self.pattern[(a,b)]
    if towrite.measurement=="In":
      try: result=result+string.lower(towrite.name)[0];
      except IndexError: result=result+"i"
    elif towrite.measurement=="Out":
      try: result=result+string.upper(towrite.name)[4]; # skipping "out("
      except IndexError: result=result+"O"
    else:
      result=result+towrite.measurement[0]
    cc=cc+1; result=result+"";
  return result+"\n";

def figobjects(self, job=["build", "place"], spacing=None, subscriptfct=lambda x: "",
               figs=None):
  """return a list of the figobjects assotiated with the measurements of
     the pattern. As a whole, the create a picture of the pattern.
     arguments are simply transferred to build_figobjects"""
  self.build_figobjects(job, spacing, subscriptfct, figs)
  return map(lambda x: x.figobject, self.pattern.values())

def build_figobjects(self, job=["build", "place"], spacing=None,
# do nothing as default job creates a mess
```

```
                  subscriptfct=lambda x: "",
                  figs=None):
"""build the figobjects for the measurements in this pattern
   arguments are: job can be "build" the whole thing or just "place" the
   existing objects, spacing set the grid, subscriptfct returns what to
   write under each icon (usually coordinates), and figs is the icon
   library to use"""

# select a small icon collection if pattern is large
if figs==None:
  ((xmin,ymin),(xmax,ymax))=self.patternsize()
  if xmax-xmin>60 or ymax-ymin>60: # guess, should fit on one page
    figs=figclasses.smallsigmas
    subscriptfct=lambda x: "" # icons are smaller than text
  else:
    figs=figclasses.sigmas

# adapt spacing automatically
if spacing==None:
  cliche=copy.deepcopy(figs["X"])
  subscript=subscriptfct((30,42))
  if subscript: cliche.addsubscript(subscript, figs["subscript"])
  ((x1,y1),(x2,y2))=cliche.size()
  (spacingx,spacingy)=((x2-x1)*1.2, (y2-y1)*1.2)
  print "AUTOMATIC spacing. size of X ", ((x1,y1),(x2,y2)), " spacing: ",\
        (spacingx,spacingy)
else:
  (spacingx,spacingy)=spacing
  print "SET spacing: ", (spacingx,spacingy)

# job contains list, what to do, if figobject exists: complete rebuild,
# place, nothing
for (x,y) in self.pattern.keys():
  cjob=job
  if not self.pattern[(x,y)].figobject: cjob=["build", "place"]
  if "build" in cjob:
    c=copy.deepcopy(figs[self.pattern[(x,y)].measurement])
    textobj=c.find_comment("# mutable") # depends on "sigmas.fig"
    if textobj: textobj.set_text(self.pattern[(x,y)].name)
    subscript=subscriptfct((x,y))
    if subscript: c.addsubscript(subscript, figs["subscript"])
    self.pattern[(x,y)].figobject=c
  if "place" in cjob:
    self.pattern[(x,y)].figobject.place((x*spacingx,
```

```
                                        y*spacingy))

def patternsize(self):
  """return size of pattern as ((xmin,ymin),(xmax,ymax))"""
  xkeys=map(lambda (x,y): x, self.pattern.keys())
  ykeys=map(lambda (x,y): y, self.pattern.keys())
  try:
    size=((reduce(min, xkeys), reduce(min, ykeys)),
          (reduce(max, xkeys), reduce(max, ykeys)))
  except TypeError: return None
  return size

def adjustsize(self):
  """corrects the size member to represent the actual dimension of the
     pattern"""
  x=0; y=0;
  for (a,b) in self.pattern.keys():
    x=max(a+1,x); y=max(b+1,y);
  self.size=(x,y);
  return self.size;

def stamp(self,other, f):
  """merges two pattern without respect to any overlay, just "prints" the
     argument over the self pattern"""
  for k in other.pattern.keys():
    self.pattern[f(k)]=copy.deepcopy(other.pattern[k]);
  return self;

def transform(self,f):
  """transform the pattern by transforming every coordinate by function
     f"""
  newpattern={}
  for (x,y) in self.pattern.keys():
    newpattern[f((x,y))]=self.pattern[(x,y)];
  self.pattern=newpattern;
  return self.pattern;

def translate(self,(x,y)):
  """returns a simple shift or move function to be used with transform"""
  return lambda (a,b),x=x,y=y: (a+x,b+y);

def flip(self,(horizontal, vertical, rightturn)):
  """returns a function, which flips or 90°-turns a coordinate system, to
     be used as transform input"""
```

```
    (x,y)=self.size; x=x-1; y=y-1;
    def fun((a,b),horizontal=horizontal, vertical=vertical,
                  rightturn=rightturn, x=x, y=y):
      if horizontal: a=x-a
      if vertical: b=x-b
      if rightturn: return (b,-a)
      else: return (a,b)
    return fun

class Gate(Pattern):
  """A measurement pattern which has a mount method to "plug in" another gate
     and a helper method mountpos"""

  def mountpos(self, other, f, joinlist=[], inouttox=0):
    """join two gates. apply transformation and check for correctness
       this is a helper function for mount"""

#    c=copy.copy(self); print "SELF:\n", c.pattern, Pattern.__str__(c)
#    c=copy.copy(other); c.transform(f); print "OTHER:\n", c.pattern, Pattern.__str__(c)
    # check distance
#    print "MOUNTPOS with joinlist ", joinlist
    for coord in other.pattern.keys():
      (x,y)=f(coord)
      for tcoord in [(x,y), (x-1,y),(x+1,y),(x,y+1),(x,y-1)]:
        if self.pattern.has_key(tcoord) and f(coord) not in joinlist \
           and tcoord not in joinlist:
           # too close
#           print "Problematic COORD ", tcoord, " original ", coord, \
#                 " transformed ", f(coord)
           raise OverlapError((tcoord,(x,y)));
    # distance is ok
    self.stamp(other, f)
    # X joinlist
    for c in joinlist:
      if self.pattern[c].measurement not in ["In", "Out"]:
        error("joinlist for mountpos should only contain pins.")
      if inouttox: self.pattern[c]=Measurement("X", "join")

    # adapt figs

  def inoutreachable(self):
    """check if all in/out-pins are reachable from the outside"""
    (xmax,ymax)=self.adjustsize()
    pins=urbild(lambda m:m.measurement in ["In","Out"], self.pattern)
```

```
    try:
      for ((x,y),v) in pins:
        Wire(self, (x,y), (xmax+5+(x+xmax)%2,ymax+5+(y+ymax)%2))
    except NoWayError:
#      print "Not all pins reachable from the outside."
      return 0
    return 1

  def namestopoints(self, other, assarray):
    """create list of points to match"""
    points=[]
#    print "MATCHPOINTS called ", \
#      map(lambda (c,m): (c,m.measurement+m.name), self.pattern.items()), \
#      map(lambda (c,m): (c,m.measurement+m.name), other.pattern.items()), \
#      assarray
    for (selfpin, otherpin) in assarray.items():
      o=inverse(lambda x, selfpin=selfpin: x.name==selfpin, self.pattern)
      p=inverse(lambda x, otherpin=otherpin: x.name==otherpin, other.pattern)
      points.append((o[0],p[0]))
    return points

  def mount(self, other, mountnames={}, inouttox=0):
    """
    join two gates. mountpoints is a assoc list of "pin" names, which must
    match each other
    """
    otherxed=copy.deepcopy(other)
    mountpoints=self.namestopoints(otherxed, mountnames)

#    print "MOUNTPOINTS ", mountpoints

    for (o,p) in mountpoints:
      # checking, that only Outputs are connected with Inputs
      pair=(self.pattern[o].measurement,
            otherxed.pattern[p].measurement)
      if not (pair==("In","Out") or pair==("Out","In")):
        error("Tried to wire pins "+str((o,p))+", which are of type "+\
              str(pair)+".")

      # replacing Out and In with X, since pins are measured to drive the
      # quantum information through the gate
      # mountpos does this

#    print points, " CREATED FROM ", items, " AND ", itemso
```

```
    # match mountpoints by selecting a transformation
    for transdesc in [(a,b,c) for c in [0,1] for b in [0,1] for a in [0,1]]:
      # f applied to other will move it to fit with self
      def f(coords, flip=self.flip(transdesc), mountpoints=mountpoints):
#        print "f RECEIVED ", coords, " POINTS ", mountpoints
        (x,y)=flip(coords)
#        print "FLIPPED ", (x,y)
        try:
          (xo,yo)=flip(mountpoints[0][1])
          a=xo-mountpoints[0][0][0] # difference of x coords of first
                                    # matching points
          b=yo-mountpoints[0][0][1] # y coords
#          print "f RETURNED ", (x+a,y+b), " WITH ", (a,b)
        except IndexError: return (x,y)
        return (x-a,y-b)
      # check whether f matches the "pins" in points list
      for i in range(0,len(mountpoints)):
        if not mountpoints[i][0]==f(mountpoints[i][1]):
#          print "TRANS", transdesc, "DID NOT MATCH POINT", i, " OF ", mountpoints
          break # try next transformation
      else:
        try:
          self.mountpos(otherxed, f, [a for (a,b) in mountpoints], inouttox)
        except OverlapError, p:
#            print "TRANS", transdesc, "OVERLAPPED", p, "WITH ", [a for (a,b) in points]
          continue # try next transformation
        return self

#    figdump(self)
#    figdump(otherxed)
    error("Cannot mount, no working transformation.")

  def matchlist(self, other, mountnames):

    mountpoints=self.namestopoints(other,mountnames)

    # create a list of pairs of matching points and transforming functions
#    print "MATCHLIST mountlist ", mountpoints
#    print "POINTS of self ", [ a for (a,b) in mountpoints], "\n", self
#    print "POINTS of other ", [ b for (a,b) in mountpoints], "\n", other
    matchlist=[]
    for ((x1,y1),(x2,y2)) in mountpoints:
      for transdesc in [(a,b,c) for c in [0,1] for b in [0,1] for a in [0,1]]:
```

```
      # rotation should be tried last, because it creates probably the most
      # wrinkled pattern (does sort respect the order?)
        def f((x,y),  flip=self.flip(transdesc),
              ((x1,y1),(x2,y2))=((x1,y1),(x2,y2))):
          (xn,yn)=flip((x,y))
          (x2f,y2f)=flip((x2,y2))
          return (xn-x2f+x1, yn-y2f+y1)
        matchlist.append(([(c1,c2) for (c1,c2) in mountpoints if c1==f(c2)], f))

    if not mountpoints:
      matchlist=[([],lambda x:x)] # if there are no mountpoints to match,
                                  # every transformation will do

    def quality((matches, f), mountpoints=mountpoints):
      # direct matches are worth 1000
      return len(matches)*1000+\
      reduce(operator.add,
             map(lambda cp, f=f: cp[0][0]-f(cp[1])[0]+cp[0][1]-f(cp[1])[1],
                 [cpair for cpair in mountpoints if cpair not in matches]),0)
             # calculate the distance of the remaining points

    matchlist.sort(lambda a,b,quality=quality: -cmp(quality(a), quality(b)))

    return matchlist

  def rubbermount(self, other, mountnames, matchlist, inouttox=0):
    """mount two gates possibly using wiring. No automatic wire crossing,
       Must be made explicit using swap gates."""

    mountpoints=self.namestopoints(other,mountnames)
###    matchlist=self.matchlist(other, mountnames)

#    print "MATCHLIST rubbermount ", matchlist
#    print "MOUNTPOINTS rubbermount ", mountpoints

    # case with old or new matchlist, to supply an old one makes
    # backtracking possible
    count=0 # just for documentation
    for (matches, f) in  matchlist:
      candidate=copy.deepcopy(self)
      count=count+1
#      print count, ".Versuch mit \nmatchlist: ", matches, "\n", candidate, \
#            "\n", repr(candidate), "\n**********************************"
      try:
```

```
        candidate.mountpos(other, f, [a for (a,b) in matches], inouttox)
      except OverlapError, p:
#        print "FUNCTION", f, "OVERLAPPED", p, "WITH ", [a for (a,b) in matches]
        continue # try next (matches, function) pair
      # wiring pins which didn't match directly
#      figdump(candidate, "/tmp/dumpcandidate.fig")
      wirecandidate=copy.deepcopy(candidate)
#      print "wirecandidate", "\n", wirecandidate, \
#            "\n", repr(wirecandidate), "\n------------------"
#      print "CANDIDATE needs WIREING for ", \
#        [(c1,f(c2)) for (c1,c2) in mountpoints if (c1,c2) not in matches]
      for (c1,c2) in [(c1,f(c2)) for (c1,c2) in mountpoints \
                      if (c1,c2) not in matches]:
        try:
          wire=Wire(wirecandidate, c1, c2)
        except NoWayError:
#          print "Cannot wire in ", wirecandidate, " from ", c1, " to ", c2
          break # try next transformation
#        print "wirecandidate.mountpos with ", [(c1,c1),(c2,c2)]
#        figdump(wire, "/tmp/dumpwire.fig")
        wirecandidate.mountpos(wire, lambda x: x, [c1,c2], inouttox)
      else: # else clause of for, executed after exhaustion of the list
        self.pattern=wirecandidate.pattern
        return self
      # continue, try next transformation
    raise NoMountError("Cannot mount, no working transformation.")

  def figcorrelations(self, xorzandpin, (spacingx, spacingy)=(600,600),
                      subscriptfct=lambda x: "",
                      figs=figclasses.correlations):
    result={}
    pattern=self.pattern.keys()
    def multiply(coord, sigma, result=result):
      try: result[coord]=result[coord]+sigma
      except KeyError: result[coord]=sigma
    for (x,y) in self.correlations[xorzandpin]:
      multiply((x,y),"X")
      multiply((x+1,y),"Z")
      multiply((x-1,y),"Z")
      multiply((x,y+1),"Z")
      multiply((x,y-1),"Z")
    figlist=[]
    for (x,y) in filter(lambda x, pattern=pattern: x in pattern,\
                        result.keys()):
```

```
      picture=result[(x,y)]
      try:
        o=figs[picture]
      except:
        o=figs["Tilted"]  # depends on "correlations.fig
      o=copy.deepcopy(o)
      textobj=o.find_comment("# mutable") # depends on "correlations.fig"
      if textobj: textobj.set_text(result[(x,y)])
      subscript=subscriptfct((x,y))
      if subscript: o.addsubscript(subscript, figs["subscript"])
      o.place((x*spacingx, y*spacingy))
      figlist.append(o)
    return figlist

  def relabel(self, rules, predhits=None):
    if predhits==None: predhits=len(rules)
    hits={}
    for old in rules.keys(): hits[old]=0
    for coord in self.pattern.keys():
      for (old,new) in rules.items():
        if self.pattern[coord].name==old:
          hits[old]=hits[old]+1
          self.pattern[coord].name=new
          break
#      print "RELABEL ",
    for old in rules.keys():
      if hits[old]==0:
        error("relabel of "+repr(self)+" with "+str(rules)+\
              " failed, because "+str((old,rules[old]))+" didn't match.")
    if not reduce(operator.add, hits.values(), 0)==predhits:
      error("relabel of "+repr(self)+" with "+str(rules)+\
            " failed, because the rules matched "+\
            str(reduce(operator.add, hits.values(), 0))+\
            " times instead of predicted "+\
            str(predhits)+".")
    return self

class Wire(Gate):
  """should create a wire avoiding the obstacles of a passed Pattern.
     Hope this works."""
  def __init__(self, ground, (x1,y1), (x2,y2)):
    Gate.__init__(self)

    def freepoint((x,y), visited,
```

```
                (x1,y1)=(x1,y1), (x2,y2)=(x2,y2), ground=ground):
      """check, whether the quantum wire can occupy a site without
         interfereing with neighouring sites, which are also occupied
         self interference is not checked, bacause it is no problem"""
      if (x,y)==(x1,y1) or (x,y)==(x2,y2): return 1
      # since start and end points are usually pins of a gate, they are
      # forced to be reachable, otherwise ... (this part is stronger than
      # the one below, since adjescant measurements are also not considered)
      for coord in [l for l in [(x,y),(x-1,y),(x+1,y),(x,y-1),(x,y+1)] \
                      if l not in [(x1,y1),(x2,y2)]]:
      # since start and end points are measurements, but the wire should
      # approach, i.e. place measurements close to them, i exclude them
        if ground.pattern.has_key(coord) or coord in visited:
#          print "FP REJECTED ", (x,y), " because of ", l, " WITH LIST ",\
#            [l for l in [(x,y),(x-1,y),(x+1,y),(x,y-1),(x,y+1)] \
#               if l not in [(x1,y1),(x2,y2)]], " STRIPPED ", [(x1,y1),(x2,y2)]
         return 0
#      print "FP ACCEPTED ", (x,y), " WITH LIST ",\
#        [l for l in [(x,y),(x-1,y),(x+1,y),(x,y-1),(x,y+1)] \
#           if l not in [(x1,y1),(x2,y2)]]
      return 1 # no conflict

    # build a quantum wire from x1,y1 to x2,y2 ovoiding obstacles indicated
    # by freepoint
    def buildline((x1,y1)=(x1,y1), (x2,y2)=(x2,y2),
                  freepoint=freepoint):
      l=[(x1,y1)]
      try: r=route((x1,y1), [], 0, (x2,y2), freepoint)
      except RuntimeError, e:
        print "Error: ", e, " let rout fail."
        return None
#      print "ROUTE ", ((x1,y1), [], 0, (x2,y2), freepoint), " RETURNS ", r
      if r:
        l.extend(r)
        return l
      else: return None

    # layout restriction imposed by the wire gate, which is two atoms long
    if not (x1-x2+y1-y2)%2==0: raise ValueError("Cannot wire even to odd.");

    # drawing wire
    bl=buildline()
#    print "BUILDLINE ", bl
    if not bl:
```

```
      raise NoWayError("Cannot wire "+str((x1,y1))+" to "+str((x2,y2))+\
        " on "+repr(ground)+".")
    for coord in bl:
      self.pattern[coord]=Measurement("X")
    # start and end points
    self.pattern[(x1,y1)]=Measurement("In", "a")
    self.pattern[(x2,y2)]=Measurement("Out", "out(a)")
    # names are necessary!

#    Gate.__init__(self, [], {});
#    for x in range(min(x1,x2),max(x1,x2)+1):
#      self.pattern[x,y2]=Measurement("X");
#    for y in range(min(y1,y2),max(y1,y2)+1):
#      self.pattern[x1,y]=Measurement("X");
#    self.adjustsize();

def read_gate(file):
  """Read in a description file named "file" creating a Gate object"""
  def lookup(m,e):
    try:
      return m[e];
    except KeyError:
      if e=="" or e==" ": return None;
      else: return e;

  fd=open(file);
  state=""
  content={"function:":[], "pattern:":[], "mapping:":[], "name:":[],
           "description:":[], "correlations:":[]}
  while 1:
    line=fd.readline();
    if not line: break;
    line=line.split("#")[0];
    if line.strip() == "": continue;
    if line.strip() in content.keys(): state=line.strip(); continue;
    content[state].append(line.split("\n")[0]);

  mapping={}
  symbolmapping={}
  for l in content["mapping:"]:
    ls=l.split("=");
    try:
      mapping[ls[0].strip()]=ls[1].strip();
      symbolmapping[ls[1].strip()]=ls[2].strip();
```

```
    except IndexError:
      pass

  sep=lambda x: list(x);
  if content["pattern:"][0].count("\t") > 0: sep=lambda x: x.split("\t");
  patternlist=map(sep, content["pattern:"]);
  for row in range(0,len(patternlist)):
    for col in range(0,len(patternlist[row])):
      e=patternlist[row][col].strip()
      if e=="": patternlist[row][col]=None; continue
      try: patternlist[row][col]=mapping[e]
      except KeyError: pass
      if not symbolmapping.has_key(e): symbolmapping[e]=e
  result=Gate(patternlist, symbolmapping);

  corr={}
  for line in content["correlations:"]:
    line=line.split()
    for e in line[1:]:
      try: corr[line[0]].append(eval(e))
      except KeyError: corr[line[0]]=[eval(e)]

  result.correlations={}
  for c in corr.keys():
    if (c[0] not in ["X","Z"]) or (not c[len(c)-1]==":"):
      error("Correlation "+c+" "+corr(c)+" from '"+file+"' isn't ok.")
    else:
      result.correlations[(c[0],c[1:len(c)-1])]=corr[c]

  result.function=content["function:"];
  if len(content["name:"])==1: result.name=content["name:"][0]
  else: raise Exception("Name of descrition file \""+fd.name+"\" ist not 1 line.")
  result.description="\n".join(content["description:"])

  fd.close();
  return result;

def posgate(gate,s):
  mygate=copy.deepcopy(gate)
  mygate.relabel({"a":"a"+s, "b":"b"+s });
  return mygate

# read an external file which creates an object and pickle it or unpickle it
```

```
def external(file):
  obj=os.path.basename(file)
  pfile=file+".pkl"
  if os.access(pfile, os.R_OK) and \
     os.stat(pfile)[stat.ST_MTIME]>=os.stat(file)[stat.ST_MTIME]:
    pf=open(pfile)
    globals()[obj]=pickle.load(pf)
    pf.close()
    print "Read "+pfile+"."
  else:
    execfile(file, globals())
    pf=open(pfile, "w")
    pickle.dump(eval(obj), pf)
    pf.close()
    print "Wrote "+pfile+"."

# read files to create a library of gates
scriptdir=os.path.abspath(sys.path[0])

wire=read_gate(scriptdir+"/wire");
swap=read_gate(scriptdir+"/swap");
fourswap=read_gate(scriptdir+"/fourswap")
#adder=read_gate(scriptdir+"/adder");
#zeroadder=read_gate(scriptdir+"/zeroadder");
#oneadder=read_gate(scriptdir+"/oneadder");
#cswap=read_gate(scriptdir+"/cswap");
cnot=read_gate(scriptdir+"/cnot");
controledphasegate=read_gate(scriptdir+"/controledphasegate");
hadamard=read_gate(scriptdir+"/hadamard");

execfile(scriptdir+"/zero")
execfile(scriptdir+"/nswap")
external(scriptdir+"/halfadder");
external(scriptdir+"/toffoli")
external(scriptdir+"/cswap")
external(scriptdir+"/carry")
external(scriptdir+"/fulladder")
external(scriptdir+"/onebitadderoc")

execfile(scriptdir+"/nbitadder", globals())
external(scriptdir+"/onebitadder")
external(scriptdir+"/fourbitadder")

external(scriptdir+"/qftnetwork");
```

```
#controledadder=copy.deepcopy(adder).mount(cswap,\
#  {"a+b+c":"a+b+c","b+0":"b", "a+0":"a"})
#twobitadder=posgate(adder,"0").mount(posgate(adder,"1"), {"C":"c"})
```

Hilfsprogramme zur Ausgabe aller für die Darstellung benötigten Bilder von Quantengattern als .fig-Dateien.

```
#!/usr/bin/python2.2

import composer, figclasses, copy, os.path, sys, re

dir="/home/bleicher/website/diplomarbeit/createdfig/"

######## create basic gates and their correlations with subscripts ##########

# import gates read(=defined) in composer

#from composer import wire,swap,\
#                     cswap, cnot, halfadder, qftnetwork,\
#                     controledphasegate, hadamard,\
#                     toffoli, carry, fulladder
## adder, zeroadder, oneadder,

#gates=[
#        "wire",
#        "hadamard",
#        "swap",
#        "cnot",
#        "controledphasegate",
#        "halfadder", "carry"
#      ]

gatefigs={}

(name, ext)=os.path.splitext(os.path.basename(sys.argv[1]))
if ext!=".fig":
  print "Extension "+ext+" of "+sys.argv[1]+" is no .fig."
  sys.exit(1)

match=re.match(r"^(?P<name>.+?)"+\
               r"((?P<meas>Correlations)(?P<cor>.)(?P<pin>.))?"+\
               r"(?P<nocoord>NC)?$", name)
```

```
if not match:
  print "File "+name+" of "+sys.argv[1]+" doesn't match."
  sys.exit(1)

gate=match.group("name")
gateo=eval("composer."+gate)

# since resizing depends on all files of a measurement/correlation group
# all files are recreated if one is needed

# from composer import eval(gate)

def labeling((x,y), nc=match.group("nocoord")):
  if nc: str((x+1,y+1))
  else: ""

# adding math mode to labels

l=filter(lambda (n,x): x[:2]!="$$", \
            [(x, "$"+x+"$") for x in map((lambda x: x.name), \
                                         gateo.pattern.values())])
gateo.relabel(dict(l), len(l))



# creating measurement pattern
gatefigs[dir+gate+".fig"]=\
                      gateo.figobjects(["build", "place"],
                      (600,800), labeling)

# calculating figure width to resize it appropriately and move them to (0,0)
((xlu,ylu),(xrl,yrl))=figclasses.figssize(gatefigs[dir+gate+".fig"])
for o in gatefigs[dir+gate+".fig"]: o.move(lambda (x,y): (x-xlu, y-ylu))
factor=xrl-xlu

#  print "Maximum width after gate "+gate+":", factor

# creating correlation figures
gateo.correlations=\
  getattr(gateo, "correlations", {})

if (match.group("meas")=="Correlations" and
    (match.group("cor"), match.group("pin")) not in
    gateo.correlations.keys()):
```

```
  print "Can not create correlation "+\
        str((match.group("cor"), match.group("pin")))+\
        " of "+name+"."
  sys.exit(1)

for (cor,pin) in gateo.correlations.keys():
  gatefigs[dir+gate+"Correlations"+cor+pin+".fig"]=\
    gateo.figcorrelations((cor,pin),\
    (600,800), labeling)

  # calculating figure width to resize it appropriately and move them to (0,0)
  ((xlu,ylu),(xrl,yrl))=figclasses.figssize(gatefigs[dir+gate+".fig"])
  for o in gatefigs[dir+gate+"Correlations"+cor+pin+".fig"]:
    o.move(lambda (x,y): (x-xlu, y-ylu))
  factor=max(factor,xrl-xlu)

#  print "Maximum width after correlation "+gate+cor+pin+":", factor

print "Overall maximal width:", factor
factor=2000.0/factor # factor was in fig resolution, with is 1200 pixels per
                     # inch
if factor>0.4: factor=0.4

print "Resulting factor:", factor

# write files

for file in gatefigs.keys():

  # resizing figure
  for o in gatefigs[file]:
    o.move(lambda (x,y), factor=factor: (factor*x,factor*y))

  # writing it to file
  fd=open(file, "w");
  figclasses.figfile.contents=gatefigs[file]
  fd.write(str(figclasses.figfile)+"\n")
  fd.close()
  print "created file "+file+"."

#!/usr/bin/python

import figclasses, sys, os.path
```

```
dir="/home/bleicher/website/diplomarbeit/createdfig/"

mname="iconMeasurement"
cname="iconCorrelation"

def measurementicon(icon):
  if icon not in [ "X", "Y", "Z", "Tilted", "In", "Out" ]:
    print "no such measurement icon: "+icon+"."
    exit(1)
  fd=open(dir+mname+icon+".fig", "w");
  figclasses.figfile.contents=[figclasses.sigmas[icon]]

  # resizing figure
  for o in figclasses.figfile.contents:
    o.move(lambda (x,y), factor=0.5: (factor*x,factor*y))

  fd.write(str(figclasses.figfile)+"\n")
  fd.close()
  print "created measurement icon "+icon+"."

def correlationicon(icon):
  if icon not in [ "X", "Z", "ZZ", "XZ", "ZX" ]:
    print "no such correlation icon: "+icon+"."
    exit(1)
  fd=open(dir+cname+icon+".fig", "w");
  figclasses.figfile.contents=[figclasses.correlations[icon]]

  # resizing figure
  for o in figclasses.figfile.contents:
    o.move(lambda (x,y), factor=0.5: (factor*x,factor*y))

  fd.write(str(figclasses.figfile)+"\n")
  fd.close()
  print "created correlation icon "+icon+"."

############################## create icons ##############################

(name, ext)=os.path.splitext(os.path.basename(sys.argv[1]))

if ext!=".fig":
  print "Extension "+ext+" of "+sys.argv[1]+" is no .fig."
  exit(1)
if name[:len(mname)]==mname:
  measurementicon(name[len(mname):])
```

```
elif name[:len(cname)]==cname:
  correlationicon(name[len(cname):])
else:
  print "No icon for "+name+"."
  exit(1)


#!/usr/bin/python

import composer, figclasses, copy

dir="/home/bleicher/website/diplomarbeit/createdfig/"

# import gates read(=defined) in composer

from composer import wire,swap,\
                     cswap, cnot, halfadder, qftnetwork,\
                     controledphasegate, hadamard,\
                     toffoli, carry, fulladder
# adder, zeroadder, oneadder,

gatefigs={}
# factor=0

for gate in [
             "wire",
             "hadamard",
             "swap",
             "cnot",
             "controledphasegate",
            ]:

  # creating measurement pattern
  gatefigs[dir+gate+".fig"]=eval(gate).figobjects(["build", "place"],
                           (600,800), lambda (x,y): str((x+1,y+1)))

  # calculating figure width to resize it appropriately and move them to (0,0)
  ((xlu,ylu),(xrl,yrl))=figclasses.figssize(gatefigs[dir+gate+".fig"])
  for o in gatefigs[dir+gate+".fig"]: o.move(lambda (x,y): (x-xlu, y-ylu))
  factor=xrl-xlu

#  print "Maximum width after gate "+gate+":", factor

  # creating correlation figures
```

```
  for (cor,pin) in eval(gate).correlations.keys():
    gatefigs[dir+gate+"Correlations"+cor+pin+".fig"]=\
      eval(gate).figcorrelations((cor,pin),\
      (600,800), lambda (x,y): str((x+1,y+1)))

    # calculating figure width to resize it appropriately and move them to (0,0)
    ((xlu,ylu),(xrl,yrl))=figclasses.figssize(gatefigs[dir+gate+".fig"])
    for o in gatefigs[dir+gate+"Correlations"+cor+pin+".fig"]:
      o.move(lambda (x,y): (x-xlu, y-ylu))
    factor=max(factor,xrl-xlu)

#  print "Maximum width after correlation "+gate+cor+pin+":", factor

print "Overall maximal width:", factor
factor=2000.0/factor # factor was in fig resolution, with is 1200 pixels per
                     # inch
print "Resulting factor:", factor

######## create basic gates and their correlations with subscripts ##########

for file in gatefigs.keys():

  # resizing figure
  for o in gatefigs[file]:
    o.move(lambda (x,y), factor=factor: (factor*x,factor*y))

  # writing it to file
  fd=open(file, "w");
  figclasses.figfile.contents=gatefigs[file]
  fd.write(str(figclasses.figfile)+"\n")
  fd.close()
  print "created file "+file+"."

############################## create icons ##############################

for measurementicon in [ "X", "Y", "Z", "Tilted", "In", "Out" ]:
  fd=open(dir+"iconMeasurement"+measurementicon+".fig", "w");
  figclasses.figfile.contents=[figclasses.sigmas[measurementicon]]

  # resizing figure
  for o in figclasses.figfile.contents:
    o.move(lambda (x,y), factor=factor: (factor*x,factor*y))

  fd.write(str(figclasses.figfile)+"\n")
```

```
  fd.close()
  print "created measurement icon "+measurementicon+"."

for correlationicon in [ "X", "Z", "ZZ", "XZ", "ZX" ]:
  fd=open(dir+"iconCorrelation"+correlationicon+".fig", "w");
  figclasses.figfile.contents=[figclasses.correlations[correlationicon]]

  # resizing figure
  for o in figclasses.figfile.contents:
    o.move(lambda (x,y), factor=factor: (factor*x,factor*y))

  fd.write(str(figclasses.figfile)+"\n")
  fd.close()
  print "created correlation icon "+correlationicon+"."

############ create measurement pattern for composed gates without #########
############ correlations

for file in ["halfadder", "carry"]:

  figs=eval(file).figobjects(["build", "place"])

#  figs=eval(file).figobjects(["build", "place"],
#                  (600,800), lambda (x,y): str((x+1,y+1)))

  # move figure to (0,0) and resize
  ((xlu,ylu),(xrl,yrl))=figclasses.figssize(figs)
  for o in figs:
    o.move(lambda (x,y), factor=factor: ((x-xlu)*factor, (y-ylu)*factor))

  # writing it to file
  filename=dir+file+".fig"
  fd=open(filename, "w");
  figclasses.figfile.contents=figs
  fd.write(str(figclasses.figfile)+"\n")
  fd.close()
  print "created file "+filename+"."

############ create full networks, filling a full page ###############

for file in ["qftnetwork",
             "toffoli",
             "cswap",
              "fulladder"]:
```

```
  figs=eval(file).figobjects()

  # calculate factor for a picture of A4 size
  ((xlu,ylu),(xrl,yrl))=figclasses.figssize(figs)
  if xrl-xlu>yrl-ylu:
    map(lambda o: o.move(lambda (x,y): (y,-x)), figs)
    ((xlu,ylu),(xrl,yrl))=figclasses.figssize(figs)
  factor=min(6000.0/(xrl-xlu),8400.0/(yrl-ylu))

  print ((xlu,ylu),(xrl,yrl)), factor

  # move figure to (0,0) and resize
  for o in figs:
    o.move(lambda (x,y), factor=factor, xlu=xlu, ylu=ylu:\
             ((x-xlu)*factor, (y-ylu)*factor))

  # writing it to file
  filename=dir+file+".fig"
  fd=open(filename, "w");
  figclasses.figfile.contents=figs
  fd.write(str(figclasses.figfile)+"\n")
  fd.close()
  print "created file "+filename+"."
```

Das Ganze wird über ein lyx-make template integriert, das eines dieser Makefiles für die zu erzeugenden Graphiken aufruft.

Im Hauptverzeichnis

```
# makefile for Dilomarbeit.lyx

.PHONY: all
all: Diplomarbeit.pdf

SOURCES= Diplomarbeit.lyx Literatur.bib code/* figures/* createdfig/*

Diplomarbeit.pdf: ${SOURCES}
# cp Diplomarbeit.pdf.bak Diplomarbeit.pdf
lyx --export pdf2 Diplomarbeit.lyx

Diplomarbeit.ps: ${SOURCES}
lyx --export ps Diplomarbeit.lyx
# perl -p -e 's/(^\\begin_inset External XFig.*\")pdf (.*\"$)/$1ps $2/' Diplomarbeit.pdf.l
Diplomarbeit.html: ${SOURCES}
```

```
cp Diplomarbeit.html.bak Diplomarbeit.html
# lyx --export html Diplomarbeit.lyx

createdfig/%:
make -C createdfig $*

image: index.html Diplomarbeit.pdf Diplomarbeit.html
#images
rm -fr image
mkdir -p image/doms/markusbleicher.de/subs/www/diplomarbeit
cp -a index.html Diplomarbeit.pdf Diplomarbeit.html \
        image/doms/markusbleicher.de/subs/www/diplomarbeit
#images

clean:
cd createdfig/; make clean
cd figures/; make clean
cd code/; make clean
rm -f Diplomarbeit.pdf
rm -f Diplomarbeit.html
rm -f Diplomarbeit.aux
rm -f Diplomarbeit.log
rm -f Diplomarbeit.tex
rm -f Diplomarbeit.toc
rm -fr image
```

In createdfig/, wo die erzeugten Gatter- und Korrelationsdarstellungen liegen

```
# makefile for Dilomarbeit/createdfig/

ICONMAKER=../code/icons.py
GATEMAKER=../code/gates.py
NETWORKMAKER=../code/networks.py

GATES=
NETWORKS=

.PRECIOUS: %.fig
.PRECIOUS: icon%.fig
.SECONDARY: halfadder.fig


icon%.fig: ${ICONMAKER}
${ICONMAKER} $@
```

```
halfadder.fig fulladder.fig qftnetwork.fig fourbitadder.fig \
carry.fig: ${NETWORKMAKER}
${NETWORKMAKER} $@

%.fig: ${GATEMAKER}
${GATEMAKER} $@

%.tex_inc: %.fig
cd ..; /usr/share/lyx/scripts/fig2psorpdftex.py createdfig/$< pdf

clean:
rm -f *.fig *.tex_inc *.eps *.pstex_t *.pdf
```

In figures/

```
# makefile for Dilomarbeit/figures

clean:
rm -f *.eps *.pdf *.tex_inc *.pstex_t
```

7.1.2 Die Gatterdateien

Ein Beispiel für eine Datei, die ein Quantengatter in ASCII-Darstellung enthält, hier das bedingte Phasengatter.

```
name:
CPG
description:
controlled phase gate, computes <00|+<01|+<10|+<11|e^i\phi.
pattern:
cX/XT
 /XZ
tX/XC
mapping:
c=c=In
C=out(c)=Out
t=t=In
T=out(t)=Out
/=$\phi$=Tilted

function:
<11|(e^i\phi-1)+1*1
correlations:
Xc: (0,0) (4,2) (1,1) (2,0)
```

```
Zc: (1,0) (2,1) (3,2)
Xt: (0,2) (4,0) (1,1) (2,2)
Zt: (1,2) (2,1) (3,0)
Xcommon: (3,0) (2,1) (3,2)
```

CNOT

```
name:
CNOT
description:
Controled Not, target is flipped if control=1 else nothing happens.
pattern:
 TXt
  X
  X
CXXXc

mapping:
X=X
C=out(c)=Out
c=c=In
T=out(t)=Out
t=t=In
function:
|1>^(c)<1|^(c)*sigma_x^(t)+|0>^(c)<0|^(c)*1^(t)
correlations:
Xt: (1,0) (3,0)
Zt: (2,0) (2,2) (3,3)
Xc: (4,3) (2,3) (0,3) (2,1) (1,0)
Zc: (3,3) (1,3)
```

Die Null
Hadamard

```
name:
hadamard
description:
Hadamard transform: |0>->|+>, |1>->|->.
pattern:
aYYYA
mapping:
a=a=In
A=out(a)=Out
function:
```

```
|+><0|+|-><1|
correlations:
Xa: (0,0) (2,0) (3,0)
Za: (1,0) (2,0) (4,0)
```

Kontrollierter Swap

```
# python script implementing a controled swap

global versions
versions=[]

recursivemount(\
  Gate(), #  empty startgate
  # list of mountjobs
    #gate to mount
  [(cnot,
    # mount names
    {},
    # relabel list
    {"c":"b", "t":"a", "out(c)":"out(b)", "out(t)":"out(a)"}),
   (toffoli,
    {"out(a)":"c_2", "out(b)":"t"},
    {"c_1":"c_temp", "out(c_1)":"out(c_temp)", "out(c_2)":"out(a)",
     "out(t)":"out(b)"}),
   (cnot,
    {"out(a)":"t","out(b)":"c"},
    {"out(t)":"out(a)","out(c)":"out(b)", "c_temp":"c",
     "out(c_temp)":"out(c)"}),
#    (copy.deepcopy(cnot).relabel({"out(c)":"out(b)"}),
#    {"out(b)":"c", "out(a)":"t"},
#    {"out(t)":"out(a)"}),
  ],
  # matchlist, to seach over working transformations (removed)
  1, # make X measurements of the In/Out Pins
  [] # [0, 2, 4] # path pointing to a specific placement and wiring
  )

cswap=versions[0][1]

figdump(cswap, "/tmp/cswap.fig")
```

Kabel

```
name:
```

```
wire
description:
Identitiy gate to build "wires" for connecting distant "pins".
pattern:
aXA
mapping:
a=a=In
A=out(a)=Out
function:
1
correlations:
Xa: (0,0) (2,0)
Za: (1,0)
```

7.1.3 programmierte Gatter

python Skripte, die Funktionen zum Erzeugen von Gatter beinhalten
  Carry

```
# python script implementing a carry gate

global versions
versions=[]

recursivemount(\
  Gate(), #  empty startgate
  # list of mountjobs
    #gate to mount
  [(toffoli,
    # mount names
    {},
    # relabel list
    {"c_1":"a", "c_2":"b", "t":"d",
     "out(c_1)":"out(a)", "out(c_2)":"out(b)", "out(t)":"out(d)"}),
   (copy.deepcopy(swap).relabel({"out(b)":"out(d)"}),
    {"out(a)":"a","out(d)":"b"},
    {}),
    (cnot, # copy.deepcopy(cnot).relabel({"out(c)":"out(b)"}),
    {"out(a)":"c", "out(b)":"t"},
    {"out(c)":"out(a)", "out(t)":"out(b)"}),
   (swap,
    {"out(a)":"a","out(b)":"b"},
    {}),
   (toffoli,
```

```
    {"out(b)":"c_2", "out(d)":"t"},
    {"c_1":"c", "out(c_1)":"out(c)", "out(c_2)":"out(b)",
     "out(t)":"out(d)"}),
  ],
  # matchlist, to seach over working transformations (removed)
  1, # make X measurements of the In/Out Pins
  [0, 12, 10, 8, 10] # path pointing to a specific placement and wiring
  )

carry=versions[0][1]

figdump(carry, "/tmp/carry.fig")
```

Kontrollierter Swap

```
# python script implementing a controled swap

global versions
versions=[]

recursivemount(\
  Gate(), #  empty startgate
  # list of mountjobs
    #gate to mount
  [(cnot,
    # mount names
    {},
    # relabel list
    {"c":"b", "t":"a", "out(c)":"out(b)", "out(t)":"out(a)"}),
   (toffoli,
    {"out(a)":"c_2", "out(b)":"t"},
    {"c_1":"c_temp", "out(c_1)":"out(c_temp)", "out(c_2)":"out(a)",
     "out(t)":"out(b)"}),
   (cnot,
    {"out(a)":"t","out(b)":"c"},
    {"out(t)":"out(a)","out(c)":"out(b)", "c_temp":"c",
     "out(c_temp)":"out(c)"}),
#    (copy.deepcopy(cnot).relabel({"out(c)":"out(b)"}),
#    {"out(b)":"c", "out(a)":"t"},
#    {"out(t)":"out(a)"}),
  ],
  # matchlist, to seach over working transformations (removed)
  1, # make X measurements of the In/Out Pins
  [] # [0, 2, 4] # path pointing to a specific placement and wiring
```

```
  )

cswap=versions[0][1]

figdump(cswap, "/tmp/cswap.fig")
```

Halbaddierer

```
# execfiled from composer, will build an halfadder from two CNOTs

global versions
versions=[]

recursivemount(\
  Gate(), #  empty startgate
  # list of mountjobs
    #gate to mount
  [(cnot,
    # mount names
    {},
    # relabel list
    {"t":"b", "out(t)":"out(b)"}),
   (copy.deepcopy(cnot).relabel({"c":"a", "out(c)":"out(a)"}),
    {"out(b)":"t"},
    {"out(t)":"out(b)"}),
  ],
  # matchlist, to seach over working transformations (removed)
  1, # make X measurements of the In/Out Pins
  [0, 2] # path pointing to a specific placement and wiring
  )

halfadder=versions[0][1]

figdump(halfadder, "/tmp/halfadder.fig")
```

Volladdierer

```
# python script implementing a full adder

# a carry gate with input d=0
carry_0=zero(copy.deepcopy(carry), "d")

# a reversed carry gate with output out(d)=0
```

```
uncarry_0=zero(copy.deepcopy(carry).relabel(\
  {"a":"out(a)", "b":"out(b)", "c":"out(c)", "d":"out(d)",
   "out(a)":"a", "out(b)":"b", "out(c)":"c", "out(d)":"d"}),
  "out(d)")
for v in uncarry_0.pattern.values():
  if v.measurement=="In": v.measurement="Out"
  if v.measurement=="Out": v.measurement="In"

figdump(carry_0, "/tmp/carry_0.fig")
figdump(uncarry_0, "/tmp/uncarry_0.fig")

uncarry_0_swapped=copy.deepcopy(uncarry_0)

global versions
versions=[]

recursivemount(uncarry_0_swapped,
  [(nswap(2).relabel({"a":"d"}),
    {"d":"out(a)", "b":"out(b)"},
    {})],
  1,
  [13],
)

print versions

uncarry_0_swapped=versions[0][1]

uncarry_0_swapped.relabel({"d":"uppercarry"})

figdump(uncarry_0_swapped, "/tmp/uncarry_0_swapped.fig")

versions=[]
recursivemount(\
  Gate(), #  empty startgate
  # list of mountjobs
    #gate to mount
  [(carry_0,
    # mount names
    {},
    # relabel list
    {}),
   (nswap(2).relabel(
    {"out(a)":"out(b)", "out(b)":"out(d)"}),
```

```
    {"out(b)":"a", "out(d)":"b"},
    {}),
   (nswap(3),
    {"out(a)":"a", "out(c)":"b", "out(b)":"c"},
    {"out(a)":"out(b)", "out(b)":"out(c)", "out(c)":"out(a)"}),
#   (nswap(2).relabel({"a":"d", "out(a)":"out(d)"}),
#    {"out(b)":"b", "out(d)":"d"},
#    {}),
   (uncarry_0_swapped,
    {"out(a)":"a","out(b)":"b", "out(c)":"c"},
    {}),
#   (Wire(Gate(),(0,0),(8,-2)), # strange case makes handwiring necessary
#    {"out(a)":"a"},
#    {}),
#   (halfadder,
#    {"out(a)":"a", "out(b)":"b", "out(c)":"c"},
#    {}),
  ],
  # matchlist, to seach over working transformations (removed)
  1, # make X measurements of the In/Out Pins
  [0, 12, 16, 16] # path pointing to a specific placement and wiring
#  None
  )

fulladder=versions[0][1]

fulladder.relabel({"uppercarry":"d"})

for (p,t) in versions:
  figdump(t, "/tmp/dumpfulladder"+str(versions.index((p,t)))+".fig")
  print "Pattern in '/tmp/dumpfulladder"+str(versions.index((p,t)))+\
        ".fig' was created with path: ", p

figdump(fulladder, "/tmp/fulladder.fig")

#figdump(halfadder, "/tmp/halfadder.fig")
```

n-Bit-Addierer

```
# include for composer: defines a function to build an n-bit-adder

def nbitadderoc(n, size): # n recursion parameter, size actual size of the
                          # whole gate counting from 0
  nr=str(size-n) # gate number
```

```
  if n==0:
    r=copy.deepcopy(onebitadderoc)
  else: # n>0
    mountnames={"c":"out(c)"}
    r=nbitadderoc(n-1,size)
#    figdump(r, "/tmp/r"+str((n-1,size))+".fig")
    r.rubbermount(onebitadderoc, mountnames,
                  r.matchlist(onebitadderoc, mountnames), 1)
#    figdump(r, "/tmp/mountedr"+str((n-1,size))+".fig")

  r.relabel({"a":"a_"+nr, "b":"b_"+nr, "d":"d_"+nr,

             "out(a)":"out(a_"+nr+")",
             "out(b)":"out(b_"+nr+")",
             "out(d)":"out(d_"+nr+")",})
  return r

def nbitadder(n):
  return zero(zero(nbitadderoc(n, n),"c"),"out(c)")
```

Toffoli

```
# creating a toffoli from CNOTs, Hadamards and CPGs

global versions
versions=[]

recursivemount(\
  Gate(), #  empty startgate
  # list of mountjobs
    #gate to mount
  [(hadamard,
    # mount names
    {},
    # relabel list
    {"a":"t", "out(a)":"out(t)"}),
   (controledphasegate,
    {"out(t)":"t"},
    {"c":"c_2", "out(c)":"out(c_2)", "out(t)":"out(t)"}),
   (copy.deepcopy(cnot).relabel({"out(t)":"out(c_2)"}),
    {"out(c_2)":"t"},
    {"c":"c_1", "out(c)":"out(c_1)"}),
   (copy.deepcopy(controledphasegate).relabel({"$\\phi$":"$-\\phi$"}, 3),
    {"out(c_2)":"c", "out(t)":"t"},
```

```
    {"out(c)":"out(c_2)"}), # out(t)=out(t)
   (swap,
    {"out(c_2)":"a","out(t)":"b"},
    {"out(a)":"out(c_2)","out(b)":"out(t)"}),
   (copy.deepcopy(cnot).relabel({"out(t)":"temp"}),
    {"out(c_2)":"t", "out(c_1)":"c"},
    {"temp":"out(c_2)", "out(c)":"out(c_1)"}),
   (swap,
    {"out(c_1)":"a","out(c_2)":"b"},
    {"out(a)":"out(c_1)","out(b)":"out(c_2)"}),
   (controledphasegate,
    {"out(t)":"t", "out(c_1)":"c"},
    {"out(c)":"out(c_1)"}),
   (hadamard,
    {"out(t)":"a"},
    {"out(a)":"out(t)"})
  ],
  # matchlist, to seach over working transformations (removed)
  1, # make X measurements of the In/Out Pins
  [0, 2, 3, 8, 0, 8, 4, 12, 0] # path pointing to a specific placement and wiring
  )

#for v in versions:
#  print "dumping version "+str(v[0])+".\n"
#  figdump(v[1], "/tmp/toffoli-version"+str(v[0])+".fig")

toffoli=versions[0][1]

#c=copy.deepcopy(toffoli)
#for x in c.pattern.values(): x.name=""

#figdump(c, "/tmp/emptytoffoli.fig")

#figdump(toffoli, "/tmp/toffoli.fig")
```

Zero

```
# a Z measurement as a 0 input

def zero(gate, pin):
  (c,m)=inverse(lambda v, pin=pin: v.name==pin, gate.pattern)
  if m.measurement not in ["In","Out"]:
    error("'zero' works only on Pins.")
```

```
  gate.pattern[c]=Measurement("Z","zero gate")
  return gate
```

7.1.4 Die figclasses zur Erzeugung von .fig-Dateien

Eine Sammlung von Klassen, mit denen sich .fig-Dateien erzeugen und manipulieren lassen.

```
# module to read, change, build .fig files
# the elements of a fig-drawing are objects of subclasses of FigObject
# a fig file can be read and a fig drawing can be written to a file
# the FigObjects can be moved and positioned
# a large part of this code is automatically extracted from the fig format
# documentation

import re, sys, copy, math, os.path

scriptpath=os.path.abspath(sys.path[0])

class FigError(Exception):
  pass

def figerror(s):
  raise FigError(s);

# calculating figure width to resize it appropriately
def figssize(listoffigs):
  return  reduce (lambda ((xlu1,ylu1),(xrl1,yrl1)),\
                          ((xlu2,ylu2),(xrl2,yrl2)):\
                         ((min(xlu1,xlu2),min(ylu1,ylu2)),\
                          (max(xrl1,xrl2),max(yrl1,yrl2))),\
                  map(lambda o: o.size(), listoffigs))

# remove comment lines from list and return them
def comment(list):
  if len(list)>0 and re.match("^\s*#",list[len(list)-1]):
    t=list.pop(); return comment(list)+[t];
  else:
    return [];

class FigFloat:
  """is a float with a precision. The constructor sets the precision
     depending on the number of figures after the decimal point."""
  def __init__(self,str="0.0"):
```

```
    try:
      (vk,nk)=str.split(".");
      self.precision=len(nk);
    except ValueError:
      self.precision=0;
    self.number=float(str);
  def __str__(self):
    return ("%."+str(self.precision)+"f")%self.number

class FigObject:
  """the base class for the zoo of elements fig drawings consist of."""
  def find_comment(self,comment):
    if self.comment==comment: return self
    if self.type==6:
      for e in self.elements:
        r=e.find_comment(comment)
        if r: return r
    return None

  def place(self, coord):
    if self.location():
#      print "LOCATION ", self.location(), "COORD ", coord
      self.move(lambda (x,y), (xt,yt)=coord, (xo,yo)=self.location():\
                  (x-xo+xt, y-yo+yt))
#    else:
#      print "NO LOCATION, NO PLACEMENT!"

class FigColor(FigObject): #    (3.1) Color Pseudo-objects (user-defined colors)
  def __init__(self,list):
    self.comment=""
    line=list.pop().split();
    self.type=int(line[0])                    # (always 0)
    self.color_number=int(line[1])            # (color number, from 32-543 (512 total))
    self.rgb=line[2]                          # (hexadecimal string describing red,
                                              #  green and blue values (e.g. #330099) )
    if (not self.type==0) or (not len(line)==3):
      figerror("Not a Color line.")

  def location(self): None
  def size(self): None

  def __str__(self):
    result=" ".join(map(str, [self.type, self.color_number, self.rgb]))
    if self.comment: result=self.comment+"\n"+result;
```

```
    return result;

class FigARC(FigObject): #    (3.2) ARC
  def __init__(self, list):
    line=list.pop().split();
    self.type=int(line[0])                  # (always 5)
    self.sub_type=int(line[1])              # (1: open ended arc
                                            #  2: pie-wedge (closed) )
    self.line_style=int(line[2])            # (enumeration type)
    self.line_thickness=int(line[3])        # (1/80 inch)
    self.pen_color=int(line[4])             # (enumeration type, pen color)
    self.fill_color=int(line[5])            # (enumeration type, fill color)
    self.depth=int(line[6])                 # (enumeration type)
    self.pen_style=int(line[7])             # (pen style, not used)
    self.area_fill=int(line[8])             # (enumeration type, -1 = no fill)
    self.style_val=FigFloat(line[9])           # (1/80 inch)
    self.cap_style=int(line[10])            # (enumeration type)
    self.direction=int(line[11])            # (0: clockwise, 1: counterclockwise)
    self.forward_arrow=int(line[12])        # (0: no forward arrow, 1: on)
    self.backward_arrow=int(line[13])       # (0: no forward arrow, 1: on)
    self.center_x=FigFloat(line[14])           # (center of the arc)
    self.center_y=FigFloat(line[15])           # (center of the arc)
    self.x1=int(line[16])                   # first point
    self.y1=int(line[17])                   # first point
    self.x2=int(line[18])                   # second point
    self.y2=int(line[19])                   # second point
    self.x3=int(line[20])                   # third point
    self.y3=int(line[21])                   # third point
    if (not self.type==5) or (not len(line)==22):
      figerror("Not a Arc line.")

    if self.forward_arrow:
      line=list.pop()
      if not line[0]=='\t':
        figerror("Missing forward arrow (ARC) line.")
      line=line[1:].split()
      self.fw_arrow_type=int(line[0])       # (enumeration type)
      self.fw_arrow_style=int(line[1])      # (enumeration type)
      self.fw_arrow_thickness=FigFloat(line[2]) # (1/80 inch)
      self.fw_arrow_height=FigFloat(line[3])    # (Fig units)
      if not len(line)==4:
        figerror("Not a forward arrow (ARC) line.")

    if self.backward_arrow:
```

```
      line=list.pop()
      if not line[0]=='\t':
        figerror("Missing backward arrow (ARC) line.")
      line=line[1:].split()
      self.bw_arrow_type=int(line[0])        # (enumeration type)
      self.bw_arrow_style=int(line[1])       # (enumeration type)
      self.bw_arrow_thickness=FigFloat(line[2]) # (1/80 inch)
      self.bw_arrow_height=FigFloat(line[3])    # (Fig units)
      if not len(line)==4:
        figerror("Not a backward arrow (ARC) line.")

  def location(self): return (self.x1, self.y1)
  def size(self): return ((min(self.x1, self.x2, self.x3), \
                           min(self.y1, self.y2, self.y3)), \
                          (max(self.x1, self.x2, self.x3), \
                           max(self.x1, self.x2, self.x3)) )

  def move(self, f):
    (self.center_x.number,self.center_x.number)\
      =f((self.center_x.number,self.center_y.number))           # (center of the arc)
    (self.x1,self.y1)=f((self.x1,self.y1))                    # first point
    (self.x2,self.y2)=f((self.x2,self.y2))                    # second point
    (self.x3,self.y3)=f((self.x3,self.y3))                    # third point
    # correcting the typing
    self.x1=int(self.x1); self.y1=int(self.y1)               # first point
    self.x2=int(self.x2); self.y2=int(self.y2)               # second point
    self.x3=int(self.x3); self.y3=int(self.y3)               # third point

  def __str__(self):
    result=" ".join(map(str, [self.type, self.sub_type, self.line_style,
    self.pen_color, self.fill_color, self.depth, self.pen_style, self.area_fill,
    self.cap_style, self.direction, self.forward_arrow,
    self.backward_arrow, self.center_x, self.center_y, self.x1, self.y1,
    self.x2, self.y2, self.x3, self.y3 ]))
    if self.forward_arrow:
      result=result+"\n\t"+" ".join(map(str, [self.fw_arrow_type, self.fw_arrow_style,
      self.fw_arrow_thickness, self.fw_arrow_height]))
    if self.backward_arrow:
      result=result+"\n\t"+" ".join(map(str, [self.bw_arrow_type,
      self.bw_arrow_style, self.bw_arrow_thickness, self.bw_arrow_height]));
    if self.comment: result=self.comment+"\n"+result;
    return result;

class FigCOMPOUND(FigObject): #    (3.3) COMPOUND
```

```
def __init__(self, list):
  line=list.pop().split();
  self.type=int(line[0])                    # (always 6)
  self.upperleft_corner_x=int(line[1])      # (Fig units)
  self.upperleft_corner_y=int(line[2])      # (Fig units)
  self.lowerright_corner_x=int(line[3])     # (Fig units)
  self.lowerright_corner_y=int(line[4])     # (Fig units)
  if (not self.type==6) or (not len(line)==5):
    figerror("Not a COMPOUND line.")
  self.elements=[]
  while (not list[len(list)-1]=="-6"):
    self.elements.append(read(list));
  list.pop(); # removing trailing "-6"

def addsubscript(self, subscript, subscriptfig):
  # subscriptfig is copied, self is changed
  t=copy.deepcopy(subscriptfig)
  baseline=t.font_size.number*1200/72 # size from baseline to baseline in
                                      # fig units
  # add text object to figobject compound and extend bounding box
  t.place((int((self.upperleft_corner_x+self.lowerright_corner_x)/2),
               self.lowerright_corner_y+int(baseline)))
  self.lowerright_corner_y=self.lowerright_corner_y+int(baseline*1.25)
  t.set_text(subscript)
  self.elements.append(t)
  return self

def location(self): return (self.upperleft_corner_x,
                            self.upperleft_corner_y)

def size(self): return ((self.upperleft_corner_x,
                         self.upperleft_corner_y),
                        (self.lowerright_corner_x,
                         self.lowerright_corner_y))

def move(self, f):
  (x1,y1)=f((self.upperleft_corner_x,self.upperleft_corner_y))
  (x2,y2)=f((self.lowerright_corner_x,self.lowerright_corner_y))
  x1=int(x1); y1=int(y1)
  x2=int(x2); y2=int(y2)
  self.upperleft_corner_x=min(x1,x2)     # (Fig units)
  self.upperleft_corner_y=min(y1,y2)     # (Fig units)
  self.lowerright_corner_x=max(x1,x2)    # (Fig units)
  self.lowerright_corner_y=max(y1,y2)    # (Fig units)
```

```
    for e in self.elements:
      e.move(f)

  def __str__(self):
    result=" ".join(map(str, [self.type, self.upperleft_corner_x,
    self.upperleft_corner_y, self.lowerright_corner_x, self.lowerright_corner_y]))+"\n"+\
    "\n".join(map(str, self.elements))+"\n-6";
    if self.comment: result=self.comment+"\n"+result;
    return result;

class FigELLIPSE(FigObject): #    (3.4) ELLIPSE
  def __init__(self,list):
    self.comment=""
    line=list.pop().split();
    self.type=int(line[0])                  # (always 1)
    self.sub_type=int(line[1])              # (1: ellipse defined by radii
                                            #  2: ellipse defined by diameters
                                            #  3: circle defined by radius
                                            #  4: circle defined by diameter)
    self.line_style=int(line[2])            # (enumeration type)
    self.thickness=int(line[3])             # (1/80 inch)
    self.pen_color=int(line[4])             # (enumeration type, pen color)
    self.fill_color=int(line[5])            # (enumeration type, fill color)
    self.depth=int(line[6])                 # (enumeration type)
    self.pen_style=int(line[7])             # (pen style, not used)
    self.area_fill=int(line[8])             # (enumeration type, -1 = no fill)
    self.style_val=FigFloat(line[9])           # (1/80 inch)
    self.direction=int(line[10])            # (always 1)
    self.angle=FigFloat(line[11])              # (radians, the angle of the x-axis)
    self.center_x=int(line[12])             # (Fig units)
    self.center_y=int(line[13])             # (Fig units)
    self.radius_x=int(line[14])             # (Fig units)
    self.radius_y=int(line[15])             # (Fig units)
    self.start_x=int(line[16])              # (Fig units; the 1st point entered)
    self.start_y=int(line[17])              # (Fig units; the 1st point entered)
    self.end_x=int(line[18])                # (Fig units; the last point entered)
    self.end_y=int(line[19])                # (Fig units; the last point entered)
    if (not self.type==1) or (not len(line)==20):
      figerror("Not a ELLIPSE line.")

  def location(self): return (self.center_x, self.center_y)

  def size(self): return ((min(self.start_x, self.end_x,\
                               self.center_x-self.radius_x),\
```

```
                            min(self.start_y, self.end_y,\
                                  self.center_y-self.radius_y)),\
                           (max(self.start_x, self.end_x,\
                                  self.center_x+self.radius_x),\
                            max(self.start_y, self.end_y,\
                                  self.center_y+self.radius_y)))

  def move(self,f):
    # origin in the center and  with the figure
    angle=self.angle.number
    xr=self.center_x+math.cos(angle)*self.radius_x
    yr=self.center_y+math.sin(angle)*self.radius_x
    (nxr,nyr)=f((xr,yr))
    xu=self.center_x-math.sin(angle)*self.radius_y
    yu=self.center_y+math.cos(angle)*self.radius_y
    (nxu,nyu)=f((xu,yu))

    (self.center_x,self.center_y)=f((self.center_x,self.center_y))
    (self.start_x,self.start_y)=f((self.start_x,self.start_y))               # (Fig units;
    (self.end_x,self.end_y)=f((self.end_x,self.end_y))                 # (Fig units; the l
    # correcting typing
    self.center_x=int(self.center_x); self.center_y=int(self.center_y)
    self.start_x=int(self.start_x); self.start_y=int(self.start_y)               # (Fig un
    self.end_x=int(self.end_x); self.end_y=int(self.end_y)                 # (Fig units; t

    # calculating the intersections of x- and y-axis of a coord system with
    self.radius_x=int(math.sqrt((nxr-self.center_x)**2+(nyr-self.center_y)**2))
    self.radius_y=int(math.sqrt((nxu-self.center_x)**2+(nyu-self.center_y)**2))
    try:
      self.angle.number=math.atan((nyr-self.center_y)/(nxr-self.center_x))\
                        -math.pi/2 # see two lines below
    except ZeroDivisionError: # x==0 means vertical
      self.angle.number=0.0 # xfig angles are counterclock and start from vertical
#      print "ZeroDivisionError in move of FigEllipse."

  def __str__(self):
    result=" ".join(map(str, [self.type, self.sub_type, self.line_style, self.thickness,
    self.pen_color, self.fill_color, self.depth, self.pen_style, self.area_fill, self.styl
    self.direction, self.angle, self.center_x, self.center_y, self.radius_x,
    self.radius_y, self.start_x, self.start_y, self.end_x, self.end_y]))
    if self.comment: result=self.comment+"\n"+result;
    return result;

class FigPOLYLINE(FigObject): #    (3.5) POLYLINE
```

```
def __init__(self,list):
  self.comment=""
  line=list.pop().split();
  self.type=int(line[0])                    # (always 2)
  self.sub_type=int(line[1])                # (1: polyline
                                            #  2: box
                                            #  3: polygon
                                            #  4: arc-box)
                                            #  5: imported-picture bounding-box)
  self.line_style=int(line[2])              # (enumeration type)
  self.thickness=int(line[3])               # (1/80 inch)
  self.pen_color=int(line[4])               # (enumeration type, pen color)
  self.fill_color=int(line[5])              # (enumeration type, fill color)
  self.depth=int(line[6])                   # (enumeration type)
  self.pen_style=int(line[7])               # (pen style, not used)
  self.area_fill=int(line[8])               # (enumeration type, -1 = no fill)
  self.style_val=FigFloat(line[9])             # (1/80 inch)
  self.join_style=int(line[10])             # (enumeration type)
  self.cap_style=int(line[11])              # (enumeration type, only used for POLYLINE)
  self.radius=int(line[12])                 # (1/80 inch, radius of arc-boxes)
  self.forward_arrow=int(line[13])          # (0: off, 1: on)
  self.backward_arrow=int(line[14])         # (0: off, 1: on)
  self.npoints=int(line[15])                # (number of points in line)
  if (not self.type==2) or (not len(line)==16):
    figerror("Not a POLYLINE line.")

  if self.forward_arrow:
    line=list.pop()
    if not line[0]=='\t':
      figerror("Missing forward arrow (POLYLINE) line.")
    line=line[1:].split()
    self.fw_arrow_type=int(line[0])         # (enumeration type)
    self.fw_arrow_style=int(line[1])        # (enumeration type)
    self.fw_arrow_thickness=FigFloat(line[2]) # (1/80 inch)
    self.fw_arrow_height=FigFloat(line[3])    # (Fig units)
    if not len(line)==4:
      figerror("Not a forward arrow (POLYLINE) line.")

  if self.backward_arrow:
    line=list.pop()
    if not line[0]=='\t':
      figerror("Missing backward arrow (POLYLINE) line.")
    line=line[1:].split()
    self.bw_arrow_type=int(line[0])         # (enumeration type)
```

```
    self.bw_arrow_style=int(line[1])       # (enumeration type)
    self.bw_arrow_thickness=FigFloat(line[2]) # (1/80 inch)
    self.bw_arrow_height=FigFloat(line[3])    # (Fig units)
    if not len(line)==4:
      figerror("Not a backward arrow (POLYLINE) line.")

                                           # (this will be the same as the 1st
                                           # point for polygon and box)
  line=""
  try:
    while list[len(list)-1][0]=='\t': line=line+" "+list.pop()[1:]
  except IndexError: pass
  line=line.split();
  self.points=[]
  for i in range(0,self.npoints):
      self.points.append((int(line[2*i]),int(line[2*i+1]))); # (x,y)

  if self.sub_type==5:
    self.pic_flipped=int(list.pop());
    self.pic_file=list.pop();
  return;

def location(self, coord): return (self.points[0][0], self.points[0][1])

def size(self):
  xmin=self.points[0][0]; ymin=self.points[0][1]
  xmax=self.points[0][0]; ymax=self.points[0][1]
  for (x,y) in self.points:
    xmin=min(x, xmin); ymin=min(y, ymin);
    xmax=max(x, xmax); ymax=max(y, ymax);
  return ((xmin,ymin),(xmax,ymax))

def move(self,f):
  for i in range(0,len(self.points)):
    (x,y)=f(self.points[i])
    self.points[i]=(int(x),int(y))

def __str__(self):
  result=" ".join(map(str, [self.type, self.sub_type, self.line_style,
  self.thickness, self.pen_color, self.fill_color, self.depth, self.pen_style,
  self.area_fill, self.style_val, self.join_style, self.cap_style, self.radius,
  self.forward_arrow, self.backward_arrow, self.npoints]))
  if self.forward_arrow:
    result=result+"\n\t"+" ".join(map(str, [self.fw_arrow_type, self.fw_arrow_style,
```

```
      self.fw_arrow_thickness, self.fw_arrow_height]))
    if self.backward_arrow:
      result=result+"\n\t"+" ".join(map(str, [self.bw_arrow_type, self.bw_arrow_style,
      self.bw_arrow_thickness, self.bw_arrow_height]))
    for i in range(0,len(self.points)):
      if i % 5 == 0: result=result+"\n\t";
      result=result+" "+str(self.points[i][0])+" "+str(self.points[i][1]);
    if self.comment: result=self.comment+"\n"+result;
    return result;

class FigSPLINE(FigObject): #    (3.6) SPLINE
  def __init__(self, list):
    line=list.pop().split();
    self.type=int(line[0])                  # (always 3)
    self.sub_type=int(line[1])              # (0: open approximated spline
                                            #      1: closed approximated spline
                                            #      2: open   interpolated spline
                                            #      3: closed interpolated spline
                                            #      4: open   x-spline
                                            #      5: closed x-spline)
    self.line_style=int(line[2])            # (See the end of this section)
    self.thickness=int(line[3])             # (1/80 inch)
    self.pen_color=int(line[4])             # (enumeration type, pen color)
    self.fill_color=int(line[5])            # (enumeration type, fill color)
    self.depth=int(line[6])                 # (enumeration type)
    self.pen_style=int(line[7])             # (pen style, not used)
    self.area_fill=int(line[8])             # (enumeration type, -1 = no fill)
    self.style_val=FigFloat(line[9])           # (1/80 inch)
    self.cap_style=int(line[10])            # (enumeration type, only used for open splin
    self.forward_arrow=int(line[11])        # (0: off, 1: on)
    self.backward_arrow=int(line[12])       # (0: off, 1: on)
    self.npoints=int(line[13])              # (number of control points in spline)
    if (not self.type==3) or (not len(line)==14):
      figerror("Not a SPLINE line.")

    if self.forward_arrow:
      line=list.pop()
      if not line[0]=='\t':
        figerror("Missing forward arrow (SPLINE) line.")
      line=line[1:].split()
      self.fw_arrow_type=int(line[0])       # (enumeration type)
      self.fw_arrow_style=int(line[1])      # (enumeration type)
      self.fw_arrow_thickness=FigFloat(line[2]) # (1/80 inch)
      self.fw_arrow_height=FigFloat(line[3])    # (Fig units)
```

```
    if not len(line)==4:
      figerror("Not a forward arrow (SPLINE) line.")

  if self.backward_arrow:
    line=list.pop()
    if not line[0]=='\t':
      figerror("Missing backward arrow (SPLINE) line.")
    line=line[1:].split()
    self.bw_arrow_type=int(line[0])        # (enumeration type)
    self.bw_arrow_style=int(line[1])       # (enumeration type)
    self.bw_arrow_thickness=FigFloat(line[2]) # (1/80 inch)
    self.bw_arrow_height=FigFloat(line[3])    # (Fig units)
    if not len(line)==4:
      figerror("Not a backward arrow (SPLINE) line.")

  line=""; sline="";
  try:
    while list[len(list)-1][0]=='\t':
      l=" "+list.pop()[1:];
      if l.count(".")==0: line=line+l;
      else: sline=sline+l;
  except IndexError: pass
  line=line.split(); sline=sline.split();
  self.points=[]
  for i in range(0,self.npoints):
    self.points.append(((int(line[2*i]),int(line[2*i+1])),\
                        FigFloat(sline[i]))); # ((x,y),spline)
  return;

def location(self, coord): return (self.points[0][0], self.points[0][1])

def size(self):
  xmin=self.points[0][0]; ymin=self.points[0][1]
  xmax=self.points[0][0]; ymax=self.points[0][1]
  for (x,y) in self.points:
    xmin=min(x, xmin); ymin=min(y, ymin);
    xmax=max(x, xmax); ymax=max(y, ymax);
  return ((xmin,ymin),(xmax,ymax))

def move(self,f):
  for i in range(0,len(self.points)):
    (p,s)=self.points[i]
    (x,y)=f(p)
    self.points[i]=((int(x),int(y)),s)
```

```
  def __str__(self):
    result=" ".join(map(str, [self.type, self.sub_type, self.line_style,
    self.thickness, self.pen_color, self.fill_color, self.depth, self.pen_style, self.area
    self.style_val, self.cap_style, self.forward_arrow, self.backward_arrow,
    self.npoints]));
    if self.forward_arrow:
      result=result+"\n\t"+" ".join(map(str, [self.fw_arrow_type, self.fw_arrow_style,
      self.fw_arrow_thickness, self.fw_arrow_height]))
    if self.backward_arrow:
      result=result+"\n\t"+" ".join(map(str, [self.bw_arrow_type, self.bw_arrow_style,
      self.bw_arrow_thickness, self.bw_arrow_height]))
    sline=""
    for i in range(0,len(self.points)):
      if i % 6 == 0: result=result+"\n\t"; sline=sline+"\n\t";
      ((x,y),s)=self.points[i]
      result=result+" "+str(x)+" "+str(y);
      sline=sline+" "+str(s);
    result=result+sline;
    if self.comment: result=self.comment+"\n"+result;
    return result;

class FigTEXT(FigObject): #    (3.7) TEXT
  def __init__(self,list):
    self.comment=""
    line=list.pop().split();
    self.type=int(line[0])                  # (always 4)
    self.sub_type=int(line[1])              # (0: Left justified
                                            #      1: Center justified
                                            #      2: Right justified)
    self.color=int(line[2])                 # (enumeration type)
    self.depth=int(line[3])                 # (enumeration type)
    self.pen_style=int(line[4])             # (enumeration , not used)
    self.font=int(line[5])                  # (enumeration type)
    self.font_size=FigFloat(line[6])          # (font size in points)
    self.angle=FigFloat(line[7])               # (radians, the angle of the text)
    self.font_flags=int(line[8])            # (bit vector)
    self.height=FigFloat(line[9])              # (Fig units)
#    if self.height.number==0: print "height IS 0 "+str(self)+" with "+repr(f)+\
#       " which maps "+str(((xlu,ylu),(xrl,yrl)))+" to "+str(((xlun,ylun),(xrln,yrln)))+".
    self.length=FigFloat(line[10])             # (Fig units)
#    if self.length.number==0: print "length IS 0 "+str(self)+" with "+repr(f)+\
#       " which maps "+str(((xlu,ylu),(xrl,yrl)))+" to "+str(((xlun,ylun),(xrln,yrln)))+".
    self.x=int(line[11])                    # (Fig units, coordinate of the origin
```

```
    self.y=int(line[12])                    #  of the string.  If sub_type = 0, it is
                                            #  the lower left corner of the string.
                                            #  If sub_type = 1, it is the lower
                                            #  center.  Otherwise it is the lower
                                            #  right corner of the string.)
    self.text=line[13]                      # (ASCII characters; starts after a blank
                                            #  character following the last number and
                                            #  ends before the sequence '\001'.  This
                                            #  sequence is not part of the string.
                                            #  Characters above octal 177 are
                                            #  represented by \xxx where xxx is the
                                            #  octal value.  This permits fig files to
                                            #  be edited with 7-bit editors and sent
                                            #  by e-mail without data loss.
                                            #  Note that the string may contain '\n'.)
    while not ("    "+self.text)[len(self.text):]=="\\001":
      self.text=self.text+"\n"+list.pop();

  def location(self): return (self.x, self.y)

  def size(self):
    # the baseline problem is not approached
    angle=self.angle.number
    upx=int(self.height.number*math.sin(angle))
    upy=int(self.height.number*math.cos(angle))
    rightx=int((1.0-self.sub_type/2.0)*self.length.number*math.cos(angle))
    righty=int((1.0-self.sub_type/2.0)*self.length.number*math.sin(angle))
    leftx=-int((self.sub_type/2.0)*self.length.number*math.cos(angle))
    lefty=-int((self.sub_type/2.0)*self.length.number*math.sin(angle))
    # point names are relative to the (possibly turned) bounding  box
    xlu=leftx+upx; ylu=lefty+upy
    xll=leftx; yll=lefty
    xru=rightx+upx; yru=righty+upy
    xrl=rightx; yrl=righty
    return ((self.x+min(xlu,xru,xll,xrl),self.y+min(ylu,yru,yll,yrl)),
            (self.x+max(xlu,xru,xll,xrl),self.y+max(ylu,yru,yll,yrl)))

  def move(self,f):
    oldheight=self.height.number
    oldlength=self.length.number

#    print "height:", self.height.number
#    print "length:", self.length.number
```

```
    angle=self.angle.number
#    if oldheight==0: print "heigth in ", self, " already 0."
    upx=int(self.height.number*math.sin(angle))
    upy=int(self.height.number*math.cos(angle))
    rightx=int((1.0-self.sub_type/2.0)*self.length.number*math.cos(angle))
    righty=int((1.0-self.sub_type/2.0)*self.length.number*math.sin(angle))
    leftx=-int((self.sub_type/2.0)*self.length.number*math.cos(angle))
    lefty=-int((self.sub_type/2.0)*self.length.number*math.sin(angle))
    # point names are relative to the (possibly turned) bounding  box
    xlu=self.x+leftx+upx; ylu=self.y+lefty+upy
    xll=self.x+leftx; yll=self.y+lefty
    xru=self.x+rightx+upx; yru=self.y+righty+upy
    xrl=self.x+rightx; yrl=self.y+righty

    (xlun,ylun)=f((xlu,ylu)); (xrln,yrln)=f((xrl,yrl))
    (xlln,ylln)=f((xll,yll)); (xrun,yrun)=f((xru,yru))

    (self.x,self.y)=f((self.x,self.y))
    self.x=int(self.x); self.y=int(self.y)

    # (rl+ll)/2-(ru+lu)/2

    upx=int((xrun+xlun)/2-(xrln+xlln)/2)
    upy=int((yrun+ylun)/2-(yrln+ylln)/2)
    rightx=xrln-self.x
    righty=yrln-self.y
    leftx=xlln-self.x
    lefty=ylln-self.y

#    if upx==0:
#      print "UPX=0 mit ", (xlun,ylun), (xrun,yrun), (xrln,yrln), (xlln,ylln)

    self.height.number=math.sqrt(upx**2+upy**2)
#    if self.height.number==0: print "height ZEROED "+str(self)+" with "+repr(f)+\
#       " which maps "+str(((xlu,ylu),(xrl,yrl)))+" to "+str(((xlun,ylun),(xrln,yrln)))+\
#       "generating (upx,upy)="+str((upx,upy))+"."
    self.length.number=math.sqrt((rightx-leftx)**2+(righty-lefty)**2)
#    if self.length.number==0: print "length ZEROED "+str(self)+" with "+repr(f)+\
#       " which maps "+str(((xlu,ylu),(xrl,yrl)))+" to "+str(((xlun,ylun),(xrln,yrln)))+".
    if upx==0: self.angle.number=cmp(upy,0)*math.pi/2-math.pi/2
    else: self.angle.number=math.atan(upy/upx)-math.pi/2
#    try:
    self.font_size.number=self.font_size.number*self.height.number/oldheight
#    except ZeroDivisionError:
```

```
#      print "ZeroDivisionError in move of FigTEXT, font_size change."

  def __str__(self):
    result=" ".join(map(str, [self.type, self.sub_type, self.color, self.depth,
    self.pen_style, self.font, self.font_size, self.angle, self.font_flags,
    self.height, self.length, self.x, self.y, self.text ]))
    if self.comment: result=self.comment+"\n"+result;
    return result;

  def set_text(self, text):
    self.text="".join(map(lambda x: (x=="\\" and "\\\\") or x, text))+"\\001"

def read(list):
  def read_type(list):
    t=list[len(list)-1].split();
    type=int(t[0]);
    return type;

  c=comment(list)
  result={0:FigColor, 5:FigARC, 1:FigELLIPSE, 2:FigPOLYLINE, 3:FigSPLINE,
          4:FigTEXT, 6:FigCOMPOUND}[read_type(list)](list);
  result.comment="\n".join(c);
  return result;

class FigFile:
  def __init__(self,file):
    # Reading Header
    fd=open(file); self.intro=fd.readlines(); fd.close();
    for i in range(0,len(self.intro)):
      l=self.intro[i]
      if l[len(l)-1]=="\n": self.intro[i]=l[:len(l)-1]
    if not self.intro[0]=="#FIG 3.2": figerror(fd.name+" not a fig file.");
    figobjects=self.intro[9:]; figobjects.reverse();
    self.intro=self.intro[:9];
    # Reading Objects
    self.contents=[];
    while len(figobjects):
      self.contents.append(read(figobjects));
  def __str__(self):
    return "\n".join(self.intro+map(str, self.contents));


# sigmas is a dictionary of icons representing single atom measurements
# visualisations of measurement patterns consist of these
```

```
sigmas={}
figfile=FigFile(scriptpath+"/sigmas.fig")
for o in figfile.contents:
  sigmas[o.comment[2:]]=o

# smallsigmas is just the same only much smaller and without text
smallsigmas={}
figfile=FigFile(scriptpath+"/smallsigmas.fig")
for o in figfile.contents:
  smallsigmas[o.comment[2:]]=o

# correlations is a dictionary of icons to show the correlations used to
# prove the correctness of a measurement pattern
correlations={}
figfile=FigFile(scriptpath+"/correlations.fig")
for o in figfile.contents:
  correlations[o.comment[2:]]=o

figfile.contents=[] # figfile is the master for building fig files
```

Literaturverzeichnis

[1] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *submitted to Physical Review A*, 1995. arXiv: quant-ph/9503016.

[2] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, ???? 1989. Bennett's trick.

[3] Jiri Blank, Pavel Exner, and Miloslav Havlicek. *Hilbert Space Operators in Quantum Physics*. AIP Press, 1994.

[4] Hans J. Briegel and Robert Raussendorf. Persistent entanglement in arrays of interacting particles. *Physical Review Letters*, 86(5):910–913, January 2001.

[5] D. E. Browne, Robert Raussendorf, and Hans J. Briegel. Quantum gates and networks in qubit lattices. 2001.

[6] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. Quantum algorithms revisited. *Phil. Trans. R. Soc. Lond. A (submitted)*, 1996.

[7] R. Cleve and J. Watrous. Fast parallel circuits for the quantum fourier transform. In *Proc. 41st Symp. on Foundations of Computer Science (2000)*, 2000.

[8] Markus Greiner, Olaf Mandel, Tilman Esslinger, Theodor W. Hänsch, and Immanuel Bloch. Quantum phase transition from a superfluid to a mott insulator in a gas of ultracold atoms. *Nature*, 6867(415):39–44, jan 2002.

[9] Jozef Gruska. *Quantum Computing*. McGraw-Hill, 1999.

[10] ??? Jaksch, Hans J. Briegel, Cirac, Gardiner, and Zoller. ??? *Physical Review Letters*, 82:1975–???, ??? 1999.

[11] Robert Raussendorf and Hans J. Briegel. A one-way quantum computer. *Physical Review Letters*, 86(22):5188–5191, May 2001.

[12] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *IEEE Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[13] A. Yao. Quantum circuit complexity. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 352–361, Los Alamitos, CA, 1993. Institute of Electrical and Electronic Engineers Computer Society Press.

Hinweis

gemäß §22 (6) der Diplomprüfungsordnung

Ich habe diese Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Markus Bleicher