Vrije Universiteit Brussel
Faculteit Wetenschappen
Departement Informatica en Toegepaste Informatica

Simulatie van Kwantumberekeningen en Toepassingen

Proefschrift ingediend met het oog op het behalen van de graad van
Licentiaat in de Toegepaste Informatica

Door: Brecht Desmet
Promotor: Prof. Dr. Theo D'Hondt
Begeleider: Ellie D'Hondt
Juni 2005

Samenvatting

We breiden de klassieke programmeertaal Lisp uit om kwantumberekeningen te kunnen beschrijven en te simuleren. De voornaamste doelstellingen bij de ontwikkeling waren het produceren van leesbare programmacode met een hoge uitdrukkingskracht en het snoeien in de exponentiële complexiteit van de kwantumberekeningen. De all-round mogelijkheden van de kwantumsimulator worden geïllustreerd door tal van populaire kwantumtoepassingen te implementeren, waaronder het algoritme van Shor. Dit algoritme kan in polynomiale rekentijd een priemfactorisatie uitrekenen.

Dankwoord

Het uitvoeren van wetenschappelijk onderzoek is zeker geen solitaire bezigheid. Een goed wetenschappelijk werk is de transitieve sluiting van verschillende kennisdomeinen en ideeën die afkomstig zijn van verschillende personen. Op deze plaats wil ik dan ook alle personen bedanken die hebben meegeparticipeerd in het tot stand komen van deze thesis.

Mijn uitzonderlijke dank gaat uit naar mijn promotor Prof. Dr. Theo D'Hondt en thesisbegeleidster Ellie D'Hondt. Haar ervaring en deskundigheid op vlak van kwantumberekeningen bezorgden deze thesis een fundamentele meerwaarde. Daarnaast gaat mijn dank uit naar Dr. Pascal Costanza voor het nalezen van de thesistekst en het formuleren van opbouwende kritiek over de programmacode van de kwantumsimulator. Als laatsten maar zeker niet als minsten wil ik alle medewerkers van het Laboratorium voor Programmeerkunde aan de Vrije Universiteit Brussel bedanken voor hun waardevol advies en bijsturingen.

Als slot wens ik graag mijn ouders, grootouders, broer, Bert Degroote en Nicolas Beckers te bedanken voor hun onconditionele steun.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Hoofdstuk 1

Introductie

Het klinkt misschien vreemd, maar de computer van morgen kan gebouwd worden binnenin een kop koffie. Dit komt omdat de cafeïnemolekule wel eens een goede basis zou kunnen zijn om een kwantumcomputer mee te bouwen. Achter het afgeslankte kwantumformaat van dergelijke computer, zit een uitzonderlijke performantie verborgen in vergelijking met klassieke computers. Deze eigenschap is al meer dan voldoende argumentatie om het idee van zo'n computer grondig te bestuderen.

1.1 Kwantumberekeningen in een notendop

Klassieke computers passen op implementatieniveau de klassieke wetten van de fysica toe. De wet van Moore geeft aan dat de snelheid van processoren om de twee jaar verdubbelt, daar ze steeds uit kleinere eenheden vervaardigd worden. Op een bepaald punt zullen de processoreenheden zodanig afgeslankt zijn, dat ze een kwantumformaat aannemen. De processoreenheden gaan zich dan gedragen volgens de wetten van de kwantummechanica.

Deze subtak van de natuurkunde beschrijft het gedrag van licht en materie op atoomniveau. De interdisciplinariteit tussen computerwetenschappen en kwantummechanica, gaf de computerwetenschappen een nieuwe dimensie. Berekeningen die gebaseerd zijn op de kwantummechanische wetten, kwantumberekeningen genaamd, beschikken over een uitzonderlijk performantievoordeel in vergelijking met klassieke computers. Dit maakt het bijvoorbeeld mogelijk om de populaire RSA-encryptie efficiënt te kraken.

1.2 Belang van interdisciplinair onderzoek

Het onderzoek rond kwantumberekeningen en -informatie is interdisciplinair. Er is een instroom van kennis uit diverse wetenschappelijke takken: informatietheorie, natuurkunde, wiskunde en computerwetenschappen. Interdisciplinariteit, ongeacht welke wetenschappen erin betrokken zijn, kunnen leiden tot *innovaties*. Het mooiste voorbeeld hiervan is de programmeertaal Smalltalk. Deze werd ontwikkeld door de bioloog Alan Kay. Hij haalde zijn inspiratie uit biologische cellen om een object-georiënteerd programmeerparadigma te ontwikkelen.

Een tweede belangrijk voordeel dat interdisciplinariteit met zich meebrengt, is de *veralgemening van bestaande theorieën*. Interdisciplinair onderzoek doet ons immers dieper nadenken over bestaande theorieën. Zonder dat er al een echte volwaardige kwantumcomputer geïmplementeerd is, kunnen we theorieën ontwikkelen die een algemener beeld van de feiten scheppen. Het mooiste voorbeeld hiervan is de these van Church-Turing die door Deutsch werd geherformuleerd. De originele these heeft een impliciete fysische aanname die onduidelijk was geformuleerd. De introductie van kwantumberekeningen verplichtte ons naar computerwetenschappen te kijken met een fysische bril. Dit stelde Deutsch in staat om een meer preciezere formulering te ontwikkelen die hij het principe van Church-Turing noemde.

1.3 Probleemstelling

In deze sectie beschrijven we twee boeiende wetenschappelijke debatten waaraan deze thesis mee participeert. Deze probleemstellingen fungeren als motivatie voor de doelstellingen uit de volgende sectie.

1.3.1 Onderzoek in de programmeerkunde

Een vraag die natuurlijk brandt op de lippen van programmeertaal designers, is of kwantumcomputers een nieuw tijdperk van programmeertalen gaan inleiden. Sinds de introductie van kwantumberekeningen in de jaren '80 wordt de programmeerkunde geconfronteerd met nieuwe uitdagingen. De kwantummechanische spelregels om berekeningen uit te voeren en de kwantum informatietheorie worden in de arena van de klassieke programmeertalen geplaatst.

Belangrijke onderzoeksvragen zijn of de huidige programmeerparadigma's voldoende krachtig zijn om de nieuwe vereisten te vervullen, of als we moeten zoeken naar nieuwe programmeerconcepten of paradigma's. We pro-

beren mee te participeren in deze wetenschappelijke dialoog door zelf een kwantumsimulator te ontwikkelen die zich onderscheid van reeds bestaande toonaangevende kwantumsimulatoren.

Het meest innovatieve aan de eigen simulator is de keuze van de programmeertaal Lisp. We gebruiken de "van beneden naar boven" programmeerfilosofie en de krachtige abstractietechnieken van Lisp om een hoog-niveau kwantum programmeertaal te ontwikkelen. Ons werk constrasteert de andere toonaangevende kwantumsimulatoren door onze multi-paradigma aanpak. We verstikken ons niet in een vooropgelegd programmeerparadigma van een specifieke programmeertaal, maar genieten van de eindeloze programmeervrijheid van Lisp. We zijn van mening dat dit nieuwe boeiende debatten zal openen in de ontwikkeling van hoog-niveau kwantum programmeertalen.

1.3.2 Onderzoek in kwantumberekeningen

Momenteel bestaan er nog geen implementaties van algemene kwantumcomputers. Deze beperking mag de wetenschap niet tegenhouden om verder te gaan met het onderzoek rond kwantumberekeningen. We verwijzen terug naar de vorige sectie en beklemtonen daarmee nogmaals het belang van interdisciplinair onderzoek.

Een kwantumsimulator is een onmisbaar instrument in deze interdisciplinaire dialoog. Het is een middel om ideeën in uit te drukken en te testen. Een kwantumsimulator gaat zelfs nog een stap verder. Hij imiteert niet alleen een kwantumcomputer, maar biedt ook de mogelijkheid om de postulaten van de kwantummechanica te overstijgen. Op die manier kunnen er experimenten uitgevoerd worden die niet gerealiseerd kunnen worden op een werkelijke kwantumcomputer. Deze manier van werken is bijzonder interessant in het onderzoek rond kwantum foutcorrectie.

1.4 Doelstelling

In deze thesis wordt de klassieke programmeertaal Lisp *uitgebreid* om met willekeurige kwantumberekeningen te kunnen omgaan. De kwantumberekeningen worden *gesimuleerd* door ze om te zetten in klassieke berekeningen. De kwantumsimulator is eigenlijk een *modellering*, omdat de simulator alle kwantumberekeningen mapt op de lineaire algebra. Het geheel moet gezien worden als *experimenteer-gereedschap*, de kwantumsimulator laat immers toe om de postulaten van de kwantummechanica te overstijgen.

De belangrijkste kwaliteitseis die we onszelf opleggen, is de *uitdrukkingskracht* van de programmacode waarmee kwantumberekeningen worden be-

schreven én gesimuleerd. Met een verhoogde uitdrukkingskracht verwijzen we impliciet naar het reduceren van code-duplicatie en het compact maken van de programmacode, dit alles om de leesbaarheid te bevorderen. Zoals reeds werd toegelicht in sectie 1.3.1 is het sleutelingrediënt, dat ons onderscheidt van bestaande kwantumsimulatoren, de keuze van de *multi-paradigma programmeertaal Lisp* en de "van beneden naar boven" programmeerfilosofie.

Een twee belangrijke kwaliteitseis van de simulatie is de efficiëntie. Als gevolg van de exponentiële complexiteit van de kwantummechanische natuur, zowel qua ruimte als tijd, zijn we verplicht om technieken in te bouwen die *snoeien in de explosieve groei*. De optimalisatie van qubits, het kwantumequivalent van de klassieke bits, situeert zich in het efficiënt voorstellen van matrices. Daarnaast zijn de kwantumberekeningen voorzien van zowel specifieke als algemene optimalisatie-algoritmes.

1.5 Overzicht

Vooraleer we de bouw van de kwantumsimulator uit de doeken doen, verdiepen we ons in een breed gamma van fundamentele theoretische kwesties binnen de kwantumberekeningen en de computerwetenschappen. We behandelen eerst twee wiskundige formalismes van kwantumberekeningen, dit zijn toestandsvectoren en densiteitsoperatoren. Deze worden in hoofdstuk 2 afgeleid uit de postulaten van de kwantummechanica. De verschillende merkwaardige kwantummechanische fenomen worden met behulp van eenvoudige voorbeelden uit de kwantumberekeningen geïllustreerd.

Hoofdstuk 3 is een computer-wetenschappelijke benadering van de kwantumberekeningen. We willen een formele definitie ontwikkelen van wat een kwantumberekening precies is en nagaan wat de fundamentele meerwaarde is van kwantumberekeningen tegenover klassieke berekeningen. Het verhaal start bij de these van Church-Turing die als eerste een klassiek berekeningsmodel introduceerde om noties als berekenbaarheid en complexiteitsgraad van problemen te vatten. We breiden deze bestaande klassieke kennis van de computerwetenschappen uit met de kwantumalternatieven, waaronder kwantum berekeningsmodellen en kwantum complexiteitsklassen.

De ontwikkelingsfilosofie van de kwantumsimulator staat gedetailleerd uiteengezet in hoofdstuk 4. Door middel van zeven objectieven proberen we een scherp beeld te vormen van welk soort kwantumsimulator we gaan ontwikkelen. De bijzondere aandacht gaat uit naar de eigenschappen van de programmeertaal Lisp en hoe deze kunnen ingezet worden in de ontwikkeling van een eigen simulator.

In de volgende twee hoofdstukken 5 en 6 wordt de ontwikkelingsfiloso-

fie geconcretiseerd. We bouwen eerst een lineaire algebra abstractielaag in hoofdstuk 5. Daarboven bouwen we een kwantum abstractielaag die volledig gebaseerd is op de lineaire algebra abstractielaag. Hoofdstuk 5 is beperkt tot de implementatie van kwantum datastructuren. Vanaf hoofdstuk 6 hebben we het over de eigenlijke simulatie van kwantumberekeningen en de efficiëntiekwesties die daarmee gepaard gaan. De gemeenschappelijke noemer van hoofdstukken 5 en 6 is het ontwikkelen van compacte en expressieve programmacode.

We brengen de kwantumberekeningen tot leven door enkele populaire kwantumtoepassingen te implementeren. Op die manier kunnen we de uitdrukkingskracht van de simulator illustreren. Elke implementatie wordt voorafgegaan door een grondige wiskundige uiteenzetting van het probleem. De keuze van kwantumtoepassingen ging uit naar het probleem van Deutsch, kwantum Fourier transformatie, het algoritme van Shör en tot slot het BB84-protocol.

Hoofdstuk 2

Postulaten van de kwantummechanica

We leiden uit de postulaten van de kwantummechanica twee formalismes van kwantumberekeningen af. De taal van toestandsvectoren is een geïdealiseerd model om pure kwantumtoestanden mee voor te stellen. Omdat dit fysisch niet exact realiseerbaar is, stellen we de taal van densiteitsoperatoren voor. Deze bundelt verschillende pure toestanden en kent er een bepaalde probabiliteit aan toe.

2.1 Wetenschappelijke dialoog

Kwantummechanica is de wetenschappelijke discipline die het gedrag van licht en materie op atoomniveau beschrijft. De belangrijkste voorvaders van deze wetenschap zijn Max Planck en Albert Einstein. Planck realiseerde zich in 1890 dat de kwantisatie van energie, het gedrag van licht zou kunnen verklaren, een in die tijd onverklaard verschijnsel. Einstein bevestigde in 1905 de theorie van Planck door toen pas ontdekte verschijnselen te verklaren, zoals het foto-elektrisch effect.

De kwantumtheorie in zijn huidige vorm kent zijn oorsprong in de jaren 1925-1930, met het baanbrekende werk van Erwin Schrödinger in de theorie van de 'golfmechanica' en Werner Heisenberg, grondlegger van de matrix-rekenmethode. Deze evoluties fungeerden als basis van de huidige wiskundige formulering van de kwantummechanica.

In 1930 publiceerde Paul Dirac in zijn werk "Principles of Quantum Mechanics"[Dir47] dat kwantummechanische systemen kunnen beschreven worden door toestandsvectoren binnen een Hilbert-ruimte. Hij introduceer-

de daarbij zijn Dirac-notatie. Dit werk werd in 1932 vervolledigd door John von Neumann in zijn boek "Mathematical Foundations of Quantum Mechanics"[vN32]. Bij von Neumann lag de klemtoon meer op een rigoureuze wiskundige formulering, in tegenstelling tot de meer pragmatische aanpak van Dirac [Uni].

Het wiskundige raamwerk van de kwantummechanica wordt beschreven door middel van *vier postulaten*. Deze beschrijven de toestandsruimte, evolutie, meting en samenstelling van kwantummechanische systemen. Richard Feynman [Fey82] stelde zich in 1982 als eerste de vraag of het mogelijk is om kwantummechanische systemen efficiënt te simuleren met klassieke computers. Zijn negatief antwoord zette David Deutsch [Deu85] aan om in 1985 een universele computer te beschrijven die opgebouwd is uit kwantummechanische elementen. Dit eerste kwantum berekeningsmodel fungeerde als brug tussen de natuurkunde (kwantummechanica) en de computerwetenschappen (kwantumberekeningen).

In de volgende secties worden begrippen relevant voor kwantumberekeningen en informatie afgeleid uit de postulaten van de kwantummechanica. Dit gebeurt door middel van twee verschillende talen: *toestandsvectoren* en *densiteitsoperatoren*. In standaard uiteenzettingen wordt de taal van toestandsvectoren gebruikt. De alternatieve aanpak met densiteitsoperatoren werd onafhankelijk voorgesteld door Landau [Lan27] en von Neumann [vN27]. Het verschil tussen beide talen wordt in de volgende secties toegelicht.

2.2 Taal van toestandsvectoren

De taal van toestandsvectoren kan uitsluitend 'ideale' geïsoleerde kwantummechanische systemen beschrijven. Onder een geïsoleerd kwantummechanisch systeem wordt begrepen dat het systeem niet interageert met andere systemen, bijvoorbeeld de omgeving. Volgens de derde hoofdwet van de thermodynamica is dergelijke constructie niet realiseerbaar, maar wel goed te benaderen. Binnen de taal van toestandsvectoren, hanteert men in de literatuur vaak de term *pure toestanden*.

2.2.1 Toestandsruimte

Het eerste postulaat van de kwantummechanica definieert de wiskundige omgeving waarin toestanden zich begeven.

Postulaat 1 *Met elk geïsoleerd fysisch systeem is een complexe vectorruimte*

$\mathbb{C}^n$ met een inwendig product[1] geassocieerd. Deze ruimte heet de toestandsruimte van het systeem. Het systeem is volledig beschreven door middel van een toestandsvector, dit is een eenheidsvector[2] in de toestandsruimte van het systeem.

Enkelvoudige qubits

Het meest eenvoudige kwantummechanisch systeem, relevant voor kwantumberekeningen en -informatie, is een *enkelvoudige qubit*. Dit is een toestandsvector gedefinieerd in een 2-dimensionale complexe vectorruimte $\mathbb{C}^2$. Beschouwen we de *rechthoekige berekeningsbasissen*[3] $|0\rangle$ en $|1\rangle$, dit zijn orthonormale basisvectoren[4] voor $\mathbb{C}^n$. Deze kan men als volgt voorstellen met matrices.

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ en } |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.1}$$

Bovenstaande vectoren gebruiken we hier[5] als de *basistoestanden* van een enkelvoudige qubit. Een willekeurige enkelvoudige qubit kan men nu definiëren als een lineaire combinatie tussen de basistoestanden $|0\rangle$ en $|1\rangle$. Dit staat weergegeven in formule (2.2). De variabelen $a, b \in \mathbb{C}$ heet men de *amplitudes*. Het postulaat geeft aan dat $|\psi\rangle$ een eenheidsvector[6] moet zijn, dit betekent dat $\||\psi\rangle\| = \sqrt{\langle\psi|\psi\rangle} = |a|^2 + |b|^2 = 1$. Men spreekt over de *normalisatie conditie* voor toestandsvectoren.

$$|\psi\rangle = a|0\rangle + b|1\rangle \tag{2.2}$$

In deze lineaire combinatie kan men het idee van *superpositie* herkennen: de qubit $|\psi\rangle$ bevindt zich in een continuüm tussen de basistoestanden $|0\rangle$ en $|1\rangle$. Als we formule (2.2) uitrekenen met matrices, krijgen we volgende kolommatrix voor de qubit $|\psi\rangle$.

$$|\psi\rangle = a|0\rangle + b|1\rangle = a\begin{pmatrix} 1 \\ 0 \end{pmatrix} + b\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} \tag{2.3}$$

[1] Men spreekt ook over een Hilbert-ruimte, zie bijlage A.1.1.

[2] Een ray wordt voorgesteld door middel van een eenheidsvector. In dit werk zijn de begrippen toestandsvector en eenheidsvector equivalent.

[3] We gebruiken de Dirac-notatie voor vectoren, zie bijlage A.1.1.

[4] De definitie van orthonormaliteit en basisvectoren staan in bijlage A.1.3.

[5] Alternatieve basistoestanden voor qubits worden behandeld in sectie 2.2.3

[6] De definitie van eenheidsvectoren staat in bijlage A.1.1.

2.2.2 Evolutie

Onderstaand postulaat beschrijft de gediscretiseerde tijdsevolutie van een geïsoleerd kwantummechanisch systeem. Dit postulaat is afgeleid van de continue versie, beschreven door de vergelijking van Schrödinger.

Postulaat 2 *De evolutie van een geïsoleerd kwantumsysteem kan beschreven worden door middel van unitaire transformaties. De toestand $|\psi\rangle$ van het systeem op tijdstip t_1 is gerelateerd met de toestand $|\psi'\rangle$ op tijdstip t_2 na toepassing van de unitaire operator U die alleen afhankelijk is van de tijdstippen t_1 en t_2.*

$$|\psi'\rangle = U|\psi\rangle \tag{2.4}$$

Kwantumoperatoren

Dit postulaat beschrijft hoe kwantummechanische systemen evolueren in functie van de tijd door het toepassen van *unitaire transformaties*. Dergelijke transformatie U van grootte n kan in de lineaire algebra beschreven worden door middel van een vierkante matrix[7] met dimensie $2^n \times 2^n$ die voldoet aan volgende voorwaarde.

$$\text{Een vierkante matrix is unitair} \Leftrightarrow U^\dagger U = UU^\dagger = I \tag{2.5}$$

De operator $U^\dagger$ noemt met de hermitiaanse[8] operator van U. Uit definitie (2.5) volgt dat het inwendig product[9] tussen $U|\psi\rangle$ en $U|\phi\rangle$ de volgende eigenschap met zich meebrengt.

$$\langle\psi|U^\dagger U|\phi\rangle = \langle\psi|I|\phi\rangle = \langle\psi|\phi\rangle \tag{2.6}$$

In de kwantumberekeningen spreekt men over het begrip *kwantumoperator* in plaats van unitaire transformatie. Het onderworpen kwantummechanisch systeem is dan een qubit. Wanneer een kwantumoperator wordt toegepast op een qubit, wordt deze operator in een enkele tijdseenheid toegepast op alle superpositietermen. Dit fenomeen heet men *kwantumparallellisme*[10]. Het exponentieel performantievoordeel van kwantumberekeningen tegenover klassieke berekeningen is daarvan een gevolg.

Zoals formule (2.4) laat blijken, vertaalt het toepassen van een kwantumoperator op een qubit zich in een matrixvermenigvuldiging. Het formaat van

[7] Bijlage A.1.4 geeft aan hoe transformaties als matrices kunnen voorgesteld worden.
[8] Definitie van hermitiaanse operatoren staat in bijlage A.1.4
[9] De definitie van het inwendig product staat in bijlage A.1.1.
[10] Dit geldt ook voor meervoudige qubits.

de kwantumoperator ($2^n \times 2^n$) en de qubit (2^n) dienen bijgevolg overeen te stemmen.

De orthonormaliteit tussen de basistoestanden van een qubit blijft bewaard doordat we alleen unitaire transformaties in beschouwing nemen. Deze voorwaarde maakt alle kwantumberekeningen *omkeerbaar*. Dit is een eigenschap waar niet alle klassieke logische poorten over beschikken.

Voorbeeld 1 *We passen de kwantumoperator X toe op de enkelvoudige qubit $|\psi\rangle = \psi_1|0\rangle + \psi_2|1\rangle$. Deze kwantumoperator is het equivalent van de klassieke negatie-operator. Merk op dat in het resultaat $|\psi'\rangle$ de rol van ψ_1 en ψ_2 werd gewisseld.*

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

$$|\psi'\rangle = X|\psi\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \psi_1 \\ \psi_2 \end{pmatrix} = \begin{pmatrix} \psi_2 \\ \psi_1 \end{pmatrix}$$

2.2.3 Meting

Vorig postulaat beschreef de unitaire evolutie van een geïsoleerd kwantummechanisch systeem. Er komt echter een moment waarop een extern fysisch systeem het kwantummechanisch systeem wil observeren. Dit laatste heet men de *meting*. Daar het systeem dan niet meer geïsoleerd is, hoeft de meetoperator niet noodzakelijkerwijs unitair te zijn.

Postulaat 3 *Kwantummetingen worden beschreven door middel van een verzameling van meetoperatoren $\{M_m\}$. Deze operatoren werken op de toestandsruimte van het gemeten systeem $|\psi\rangle$. De index m verwijst naar de mogelijke meetresultaten. Elke mogelijke meetwaarde wordt met een bepaalde kans geassocieerd. Deze wordt gegeven door de probabiliteitsfunctie*

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle \tag{2.7}$$

De toestand van het systeem na de meting ziet er dan als volgt uit:

$$|\psi'\rangle = \frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}} \tag{2.8}$$

Alle meetoperatoren dienen aan de volledigsvergelijking te voldoen:

$$\sum_m M_m^\dagger M_m = I \tag{2.9}$$

von Neumann meten van een enkelvoudige qubit

Gedurende de levensduur van een willekeurige qubit $|\psi\rangle = a|0\rangle + b|0\rangle$ kan zijn toestand (dit zijn de amplitudes a en b) niet zonder effect geraadpleegd worden. Wanneer een extern fysisch systeem de qubit observeert, vervalt de qubit $|\psi\rangle$ onherroepelijk in één van zijn mogelijke meetwaardes m.

Met elke mogelijke meetwaarde m is een probabiliteit $p(m)$ geassocieerd. De normalisatie conditie voor eenheidsvectoren zorgt ervoor dat de som van de probabiliteiten 1 is. Daar enkel unitaire transformaties worden toegepast op qubits, blijft deze voorwaarde gevrijwaard. In de rest van deze sectie verklaren we algebraïsch hoe een meting verloopt.

Beschouwen we de von Neumann meetoperatoren $M_0 = |0\rangle\langle 0|$ en $M_1 = |1\rangle\langle 1|$ die inwerken op enkelvoudige qubits. Men kan verifiëren dat M_0 en M_1 voldoen aan de volledigheidsvergelijking (2.9).

$$M_0 = |0\rangle\langle 0| = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$M_1 = |1\rangle\langle 1| = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

$$M_0 + M_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$$

De kans geassocieerd met elk meetresultaat m wordt beschreven in formule (2.7). Toegepast op meetwaarde $m = 0$ (dit komt overeen met $|0\rangle$), geeft deze formule:

$$p(0) = \langle\psi|M_0^\dagger M_0|\psi\rangle = \langle\psi|M_0|\psi\rangle = |a|^2 \tag{2.10}$$

Op gelijkaardige wijze kan de kans voor $m = 1$ (dit komt overeen met $|1\rangle$) berekend worden.

$$p(1) = \langle\psi|M_1^\dagger M_1|\psi\rangle = \langle\psi|M_1|\psi\rangle = |b|^2 \tag{2.11}$$

Omdat $|\psi\rangle$ een eenheidsvector is, geldt dat $\||\psi\rangle\| = \sqrt{\langle\psi|\psi\rangle} = |a|^2+|b|^2 = 1$. Met andere woorden, de som van de probabiliteiten uit formules (2.10) en (2.10) is 1.

Voorbeeld 2 *Beschouwen we een qubit $|\psi\rangle$ die geprepareerd is in de begintoestand $1|0\rangle + 0|1\rangle$. We passen daarop de Hadamard kwantumoperator toe.*

$$|\psi'\rangle = H|\psi\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

De probabiliteitsfunctie geeft aan dat $p(0) = \frac{1}{2}$ *en* $p(1) = \frac{1}{2}$*. De mogelijke meetresultaten van* $|\psi'\rangle$ *zien er als volgt uit door toepassing van formule (2.8).*

$$|\psi''\rangle = 1|0\rangle + 0|1\rangle = |0\rangle \vee |\psi''\rangle = 0|0\rangle + 1|1\rangle = |1\rangle$$

Meting met alternatieve berekeningsbasissen

Naast de vaak gebruikte rechthoekige berekeningsbasissen $|0\rangle$ en $|1\rangle$, zijn de willekeurige berekeningsbasissen $|a\rangle$ en $|b\rangle$ ook mogelijk zolang deze maar orthonormaal zijn. Dit laatste betekent dat $\langle a|b\rangle = 0$ en dat $\||a\rangle\| = \||b\rangle\| = 1$. Bij wijze van voorbeeld introduceren we de *diagonale berekeningsbasissen* $|+\rangle$ en $|-\rangle$. Zoals aangeven in formules (2.12) en (2.13), kan dit paar berekeningsbasissen uitgedrukt worden in termen van $|0\rangle$ en $|1\rangle$.

$$|+\rangle = H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.12}$$

$$|-\rangle = H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.13}$$

De *manier* waarop de geassocieerde kans van een basistoestand wordt berekend, blijft gelijk voor willekeurige berekeningsbasissen $|a\rangle$ en $|b\rangle$. Het *meetresultaat* hangt wel af van de gebruikte berekeningsbasissen. We verduidelijken deze kwestie door de probabiliteiten te berekenen voor de qubit $|\psi\rangle$ in de rechthoekige en diagonale berekeningsbasissen.

$$|\psi\rangle = a|0\rangle + b|1\rangle = a\frac{|0\rangle + |1\rangle}{\sqrt{2}} + b\frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{a+b}{\sqrt{2}}|+\rangle + \frac{a-b}{\sqrt{2}}|-\rangle \tag{2.14}$$

Het meten van de qubit $|\psi\rangle$ in de diagonale berekeningsbasissen $|+\rangle$ en $|-\rangle$, geeft ons de respectievelijke probabiliteiten (2.15) en (2.16).

$$p(+) = \frac{|a+b|^2}{2} \tag{2.15}$$

$$p(-) = \frac{|a-b|^2}{2} \tag{2.16}$$

Wanneer we de qubit $|\psi\rangle$ meten in de rechthoekige berekeningsbasissen $|0\rangle$ en $|1\rangle$, zien de respectievelijke probabiliteiten (2.17) en (2.18) er anders uit.

$$p(0) = |a|^2 \tag{2.17}$$

$$p(1) = |b|^2 \tag{2.18}$$

In het meest algemene geval, moet bij een meting steeds de berekeningsbasissen $|a\rangle$ en $|b\rangle$ vermeld worden. Indien de berekeningsbasissen niet expliciet vermeld zijn, nemen we aan dat een meting steeds in $|0\rangle$ en $|1\rangle$ gebeurd.

Het wisselen van berekeningsbasissen bij constructie en meting, wordt binnen de kwantumberekeningen creatief ingezet om veilige encryptieprotocollen te ontwikkelen. Dit staat behandeld in sectie 7.4.

2.2.4 Samengestelde systemen

Het volgende postulaat geeft aan dat het tensorproduct[11] als wiskundige structuur wordt ingezet om samengestelde kwantummechanische systemen te construeren. Op die manier kunnen meervoudige qubits en kwantumoperatoren gedefinieerd worden.

Postulaat 4 *De toestandsruimte van een samengesteld fysisch systeem is het tensorproduct van de toestandsruimtes van het samengesteld systeem. Met andere woorden, beschouwen we n systemen $|\phi_i\rangle$ met $i \in \{1 \dots n\}$, dan wordt het volledige systeem beschreven door $|\phi_1\rangle \otimes |\phi_2\rangle \otimes \dots \otimes |\phi_n\rangle$.*

Meervoudige qubits

De complexe vectorruimte $\mathbb{C}^2$ uit sectie 2.2.1 kan uitgebreid worden door middel van het tensorproduct om kwantummechanische systemen te construeren die bestaan uit meerdere partikels. Op die manier kunnen meervoudige qubits gedefinieerd worden. Beschouwen we twee enkelvoudige qubits $|\psi\rangle \in \mathbb{C}^2$ en $|\phi\rangle \in \mathbb{C}^2$, daaruit volgt dat $|\psi\rangle \otimes |\phi\rangle = |\psi\rangle|\phi\rangle = |\psi, \phi\rangle = |\psi\phi\rangle \in \mathbb{C}^4$.

Daar $\dim(\mathbb{C}^n \otimes \mathbb{C}^m) = \dim(\mathbb{C}^n) \times \dim(\mathbb{C}^m)$, zal het aantal amplitudes in de matrixvoorstelling van deze meervoudige qubits exponentieel $f(n) = 2^n$ toenemen. We illustreren dit met een eenvoudig voorbeeld. Nemen we $\psi_1, \psi_2, \phi_1, \phi_2 \in \mathbb{C}$, dan geldt:

[11] De definitie en eigenschappen van het tensorproduct staan in bijlage A.1.5.

$$|\psi\rangle \otimes |\phi\rangle = \begin{pmatrix} \psi_1 \\ \psi_2 \end{pmatrix} \otimes \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} = \begin{pmatrix} \psi_1\phi_1 \\ \psi_1\phi_2 \\ \psi_2\phi_1 \\ \psi_2\phi_2 \end{pmatrix} \tag{2.19}$$

Nemen we nu $a = \psi_1\phi_1, b = \psi_1\phi_2, c = \psi_2\phi_1$ en $d = \psi_2\phi_2$, dan kunnen we de voorgaande matrixvoorstelling herformuleren in de volgende expressie.

$$|\psi\rangle \otimes |\phi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle \tag{2.20}$$

Merk op dat de superpositie tussen de basistoestanden $|0\rangle$ en $|1\rangle$ uit formule (2.2) op natuurlijke wijze is uitgebreid naar de basistoestanden $|00\rangle$, $|01\rangle$, $|10\rangle$ en $|11\rangle$. Deze fungeren als orthonormale basisvectoren voor $\mathbb{C}^4$. Bovenstaande expressie zullen we verder een 2-qubit noemen. Op gelijkaardige wijze kunnen willekeurige n-qubits $|\psi\rangle$ geconstrueerd worden. De normalisatie conditie geeft aan dat $\sum_{i=0}^{2^n-1} |a_i|^2 = 1$ voor alle amplitudes a_i.

$$|\psi\rangle = \sum_{i=0}^{2^n-1} a_i|i\rangle \tag{2.21}$$

Verstrengeling van qubits

Beschouwen we volgende 2-qubits die we *Bell toestanden* of *EPR-paren* noemen. Deze laatste naam is afkomstig van de wetenschappers Einstein, Podolsky en Rosen die als eerste de merkwaardige eigenschappen van deze qubitparen vaststelden.

$$|\beta_{00}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2.22}$$

$$|\beta_{01}\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} \tag{2.23}$$

$$|\beta_{10}\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}} \tag{2.24}$$

$$|\beta_{11}\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \tag{2.25}$$

Bovenstaande qubits zijn *niet scheidbaar* als een tensorproduct van enkelvoudige qubits. Concreet betekent dit dat je geen $|\phi_1\rangle$ en $|\phi_2\rangle$ kan specificeren zodanig dat $|\beta_{00}\rangle = |\phi_1\rangle \otimes |\phi_2\rangle$. De verklaring hiervoor is dat de qubits $|\beta_{00}\rangle$, $|\beta_{01}\rangle$, $|\beta_{10}\rangle$ en $|\beta_{11}\rangle$ *verstrengeld* zijn.

De merkwaardige eigenschap die verstrengeling met zich meebrengt, is de *correlatie* tussen het meten van een qubit en het meetresultaat van andere qubits die deel uitmaken van eenzelfde kwantumtoestand.

Voorbeeld 3 *Veronderstel dat we de eerste qubit van het EPR-paar β_{00} meten. De toestand na meting is $|0\rangle$ of $|1\rangle$, beide met kans $p = 1/2$. We kunnen nu op basis van het meetresultaat van de eerste meting, het meetresultaat van een tweede meting voorspellen.*

Men kan een onderscheid maken tussen *drie graderingen* van verstrengeling:

- *Geen verstrengeling.* De qubit kan beschreven worden als een tensorproduct van enkelvoudige qubits.
- *Partiële verstrengeling.* Bijvoorbeeld $\frac{|001\rangle+|010\rangle}{\sqrt{2}} = |0\rangle|\beta_{00}\rangle$ om dat we hier de eerste qubit kunnen afzonderen van de rest.
- *Totale verstrengeling.* Bijvoorbeeld de EPR-paren uit formules (2.22), (2.23), (2.24) en (2.25).

Toestandsklassen van meervoudige qubits

De verschillende toestanden waarin qubits zich bevinden, kan men als volgt samenvatten. Beschouwen we een willekeurige n-qubit $|\psi\rangle$ met amplitudes a_i, $0 \leq i < 2^n$.

Basistoestanden Λ Wanneer er een amplitude $a_j = 1$ is en de rest 0, gedraagt de qubit $|\psi\rangle$ zich als klassieke bits. Bijvoorbeeld $|0\rangle$ of $|1\rangle$.

Superposities Υ Wanneer er twee of meer amplitudes $a_j \neq 0$, is er sprake van een superpositie. Dit betekent dat de qubit $|\psi\rangle$ zich tegelijkertijd in verschillende basistoestanden bevindt. Het aantal mogelijk superposities is oneindig.

Scheidbare superposities Π Wanneer de qubit $|\psi\rangle$ kan geformuleerd worden als het tensorproduct n enkelvoudige qubits, is $|\psi\rangle$ scheidbaar.

Verstrengelde superposities Γ Alle qubits die niet scheidbaar zijn, noemt men verstrengeld $\Upsilon = \Gamma + \Pi$.

Meervoudige kwantumoperatoren

Enkelvoudige kwantumoperatoren $U_1, U_2, \ldots, U_n$ kunnen eveneens uitgebreid worden door middel van het tensorproduct $U_1 \otimes U_2 \otimes \ldots \otimes U_n$. Wanneer een kwantumoperator U wordt toegepast op een n-qubit $|\psi\rangle$, moet de grootte $2^n \times 2^n$ van de matrix U overeenstemmen met het aantal amplitudes 2^n van de n-qubit $|\psi\rangle$ om een matrixvermenigvuldiging te kunnen uitvoeren. We illustreren dit idee in onderstaand voorbeeld.

Voorbeeld 4 *Beschouwen we een 5-qubit $|\psi\rangle$. We passen de Hadamard kwantumpoort H toe op de eerste en derde qubit. We kunnen het resultaat $|\psi'\rangle$ als volgt formuleren.*

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

$$|\psi'\rangle = (H \otimes I \otimes H \otimes I \otimes I \otimes I)|\psi\rangle$$

De niet-klonen stelling

Daar kwantumoperatoren kunnen inwerken op meervoudige qubits, rijst de vraag of er een kwantumoperator U bestaat die willekeurige n-qubits kan kopiëren. Men kan bewijzen dat dergelijke algemene kwantumoperator U niet bestaat.

Het bewijs gaat als volgt. Beschouwen we een kwantummachine die bestaat uit een bronqubit $|\psi\rangle = a|0\rangle + b|1\rangle$ en een doelqubit $|\phi\rangle$. De bedoeling is om de toestand van de bronqubits weg te schrijven in de doelqubit. Veronderstellen we daarvoor een denkbeeldige kwantumoperator U die de kopieerprocedure realiseert. Idealiter gedraagt deze kwantumoperator zich als volgt:

$$\begin{aligned} U(|\psi\rangle \otimes |\phi\rangle) &= |\psi\rangle \otimes |\psi\rangle \\ &= (a|0\rangle + b|1\rangle) \otimes (a|0\rangle + b|1\rangle) \\ &= a^2|0\rangle|0\rangle + ab|0\rangle|1\rangle + ba|1\rangle|0\rangle + b^2|1\rangle|1\rangle \end{aligned} \tag{2.26}$$

De kwantumoperator U moet ook van toepassing zijn op de basistoestanden $|0\rangle$ en $|1\rangle$. Dit wordt weergegeven in de formules (2.27) en (2.28).

$$U(|0\rangle \otimes |\phi\rangle) = |0\rangle \otimes |0\rangle \tag{2.27}$$

$$U(|1\rangle \otimes |\phi\rangle) = |1\rangle \otimes |1\rangle \tag{2.28}$$

De lineariteit van de kwantumoperator U impliceert dat:

$$\begin{aligned} U(|\psi\rangle \otimes |\phi\rangle) &= U((a|0\rangle + b|1\rangle) \otimes \phi) \\ &= a|0\rangle|0\rangle + b|1\rangle|1\rangle \end{aligned} \tag{2.29}$$

Omdat de formules (2.26) en (2.29) niet altijd gelijk zijn, volgt dat er geen algemene kwantumoperator U bestaat zodat $U(|\psi\rangle \otimes |\phi\rangle) = |\psi\rangle \otimes |\psi\rangle$.

2.3 Taal van densiteitsoperatoren

In sectie 2.2 werd reeds toegelicht dat volledig geïsoleerde kwantummechanische systemen niet realiseerbaar zijn. Als we een systeem volledig willen beschrijven, moeten we ook de omgeving in beschouwing nemen. De volledige toestandsruimte $\mathbb{C}_t^n$ is de samenstelling van een systeem $\mathbb{C}_s^{n_1}$ met zijn omgeving $\mathbb{C}_o^{n_2}$.

$$\mathbb{C}_t^n = \mathbb{C}_s^{n_1} \otimes \mathbb{C}_o^{n_2} \tag{2.30}$$

Het is niet mogelijk om de ruimte $\mathbb{C}_t^n$ te karakteriseren daar deze afhankelijk is van een particuliere omgeving. Bijvoorbeeld, een experimentele kwantumcomputer die verstrengeld raakt met zijn experimenteeromgeving. Er is dus sprake van een gebrek aan informatie. De taal van densiteitsoperatoren weet dit probleem aan te pakken door de introductie van gemengde toestanden. Dit concept wordt in de volgende sectie toegelicht.

2.3.1 Densiteitsoperator

De taal van densiteitsoperatoren laat toe een kwantummechanisch systeem ρ te beschrijven waarvan de toestand niet volledig gekend is. Veronderstellen we dat ρ zich in de pure toestanden $|\psi_i\rangle$ kan bevinden met elk een geassocieerde kans $p_i \in \mathbb{R} : 0 < p_i < 1$. Verder is $\sum_i p_i = 1$.

De paren $\{|\psi_i\rangle, p_i\}$ noemt men *gemengde toestanden*. Deze probabilistische aanpak lost het probleem van gebrek aan informatie op. Op basis van deze paren kan de *densiteitsoperator* of densiteitsmatrix als volgt gedefinieerd worden.

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| \tag{2.31}$$

Merk op dat de taal van toestandsvectoren uitsluitend pure toestanden kan voorstellen. Elke pure toestand $|\psi\rangle$ kan voorgesteld worden door een densiteitsoperator $\rho = |\psi\rangle\langle\psi|$. Het omgekeerde is niet alijd geldig.

2.3.2 Herformulering van de postulaten

Toestandsruimte

Het eerste postulaat dat de toestandsruimte definieert, is ruimer dan in de taal van toestandsvectoren omdat, naast pure toestanden, gemengde toestanden hier ook mogelijk zijn.

Postulaat 1 *Met elk geïsoleerd fysisch systeem is een complexe vectorruimte met inwendig product geassocieerd (d.i. Hilbert ruimte). Deze ruimte heet de toestandsruimte van het systeem. Het systeem is volledig beschreven door middel van een densiteitsoperator, dit is een positieve operator ρ met spoor*[12] *1 die inwerkt op de toestandsruimte van het systeem. Wanneer een kwantumsysteem zich in de toestand ρ_i bevindt met kans p_i, dan is de densiteitsoperator voor het systeem $\sum_i p_i\rho_i$.*

Evolutie

De evolutie gebeurt hier eveneens door middel van unitaire transformaties. De transformatie in kwestie wordt toegepast op alle paren $\{|\psi_i\rangle, p_i\}$ van de gemengde toestand. De kans p_i blijft steeds ongewijzigd.

Postulaat 2 *De evolutie van een geïsoleerd kwantumsysteem kan beschreven worden door middel van unitaire transformaties. De toestand ρ van het systeem op tijdstip t_1 is gerelateerd met de toestand ρ' op tijdstip t_2 na toepassing van de unitaire operator U die alleen afhankelijk is van de tijdstippen t_1 en t_2.*

$$\rho' = \sum_i p_i U|\psi_i\rangle\langle\psi_i|U^\dagger = U\rho U^\dagger \tag{2.32}$$

Meting

De kans p_i geassocieerd met alle mogelijke toestanden $|\psi_i\rangle$ moet tijdens de meting ook in rekening gebracht worden. De formule uit onderstaand postulaat 3 kan volledig afgeleid worden uit formule (2.7). Eerst berekenen we de kans op meetresultaat m, gegeven dat toestand $|\psi_i\rangle$ van toepassing is.

[12] De definitie van de spooroperator staat in bijlage A.2.1.

$$p(m|i) = \langle\psi_i|M_m^\dagger M_m|\psi_i\rangle = \mathrm{tr}(M_m^\dagger M_m|\psi_i\rangle\langle\psi_i|)$$

De rekenregels van de spoor-operator en de merkwaardige overgang van inwendig product naar spoor-operator staan uiteengezet in bijlage A.2. Volgens de wet van totale probabiliteit, kunnen we nu $p(m)$ berekenen.

$$\begin{aligned} p(m) &= \sum_i p(m|i)p_i \\ &= \sum_i p_i \,\mathrm{tr}(M_m^\dagger M_m|\psi_i\rangle\langle\psi_i|) \\ &= \mathrm{tr}(M_m^\dagger M_m \sum_i p_i|\psi_i\rangle\langle\psi_i|) \\ &= \mathrm{tr}(M_m^\dagger M_m\rho) \end{aligned}$$

Na meting met meetresultaat m zullen de paren van de gemengde toestand er als volgt uitzien $\{|\psi_i^m\rangle, p(i|m)\}$. Na toepassing van formule (2.31), krijgen we volgende densiteitsoperator.

$$\begin{aligned} \rho' &= \sum_i p(i|m)|\psi_i^m\rangle\langle\psi_i^m| \\ &= \sum_i p(i|m)\frac{M_m|\psi_i\rangle\langle\psi_i|M_m^\dagger}{\langle\psi_i|M_m^\dagger M_m|\psi_i\rangle} \\ &= \sum_i \frac{p_i p(m|i)}{p(m)}\frac{M_m|\psi_i\rangle\langle\psi_i|M_m^\dagger}{\mathrm{tr}(M_m^\dagger M_m|\psi_i\rangle\langle\psi_i|)} \\ &= \sum_i p_i\frac{M_m|\psi_i\rangle\langle\psi_i|M_m^\dagger}{\mathrm{tr}(M_m^\dagger M_m\rho)} \\ &= \frac{M_m\sum_i(p_i|\psi_i\rangle\langle\psi_i|)\,M_m^\dagger}{\mathrm{tr}(M_m^\dagger M_m\rho)} \\ &= \frac{M_m\rho M_m^\dagger}{\mathrm{tr}(M_m^\dagger M_m\rho)} \end{aligned}$$

Postulaat 3 *Kwantummetingen worden beschreven door middel van een verzameling van meetoperatoren $\{M_m\}$. Deze operatoren worden toegepast op de toestandsruimte van het gemeten systeem ρ. De index m verwijst naar de mogelijke meetresultaten. Elke mogelijke meetwaarde wordt met een bepaalde*

kans geassocieerd. Deze wordt beschreven door middel van de probabiliteitsfunctie

$$p(m) = \mathrm{tr}(M_m^\dagger M_m \rho) \tag{2.33}$$

De toestand van het systeem na de meting ziet er dan als volgt uit:

$$\rho' = \frac{M_m \rho M_m^\dagger}{\mathrm{tr}(M_m^\dagger M_m \rho)} \tag{2.34}$$

Alle meetoperatoren dienen aan de volledigheidsvergelijking te voldoen:

$$\sum_m M_m^\dagger M_m = I \tag{2.35}$$

Samengestelde systemen

Densiteitsoperatoren kunnen samengesteld worden door middel van het tensorproduct.

Postulaat 4 *De toestandsruimte van een samengesteld fysisch systeem is het tensorproduct van de toestandsruimtes van het samengesteld systeem. Beschouwen we n systemen ρ_i met $i \in \{1 \ldots n\}$, dan wordt het volledige systeem beschreven door $\rho_1 \otimes \rho_2 \otimes \ldots \otimes \rho_n$.*

Hoofdstuk 3

Fundamenten van kwantumcomputers

In dit hoofdstuk onderzoeken we de fundamentele meerwaarde van kwantumcomputers in vergelijking met klassieke computers. We gebruiken het formalisme van de klassieke Turing machine om aan te tonen dat de kwantum Turing machine exact dezelfde klasse van functies berekent. Daarnaast wordt aangetoond vanwaar het exponentieel performantie-voordeel van de kwantum Turing machine komt. Tot slot introduceren we het kwantum circuit-model als equivalent model van de kwantum Turing machine om kwantumberekeningen in uit te drukken.

3.1 Wetenschappelijke dialoog

In deze sectie geven we een korte terugblik op de fundamenten van de klassieke computerwetenschappen. We volgen het principe van Church-Turing, wat zich onderscheidt van de these van Church-Turing, als motivering voor een universele kwantumcomputer.

3.1.1 These van Church-Turing

De theorie over berekenbaarheid binnen de computerwetenschappen steunt op de *these van Church-Turing*. Deze twee wetenschappers [Chu36], [Tur36] introduceerden in 1936 elk een verschillend berekeningsmodel (respectievelijk lambda-calculus en Turing machine) om het intuïtieve begrip *algoritme* en *berekenbaarheid* wiskundig te kunnen definiëren. Hun resultaten waren equivalent, ze concludeerden samen:

> "The class of functions computable by a Turing machine corresponds exactly to the class of functions which we would naturally regard as being computable by an algorithm."
>
> Alan Turing, 1936

De these geeft de equivalentie weer tussen een *wiskundige definitie* van berekenbaarheid en het *intuïtieve idee* van berekenbaarheid. Met andere woorden, elk berekenbaar probleem kan geformuleerd worden in termen van een Turing machine. Daar een Turing machine de volledig notie van berekenbaarheid dekt, impliceert dit dat elk (voldoende rijk) berekeningsmodel *gesimuleerd* kan worden door middel van een Turing machine. Alle berekeningsmodellen zijn dus equivalent. De these van Church-Turing kon tot op heden niet ontkracht worden.

In dit werk gebruiken we de these van Church-Turing in sectie 3.2.2 om aan te tonen dat een kwantumcomputer exact dezelfde klasse van functies berekent als een klassieke computer.

3.1.2 Principe van Church-Turing

De fysicus David Deutsch [Deu85] gaf aan dat er een impliciet fysisch principe verborgen zit in de these van Church-Turing. De Turing machine is gebaseerd op de klassieke wetten van de fysica. Concreet betekent dit, bijvoorbeeld, dat de toestanden op het bandgeheugen steeds over lees- en schrijfmogelijkheden bezitten of dat toestanden ten alle tijde gekopieerd kunnen worden. Merk op dat dergelijke eigenschappen binnen de kwantumberekeningen niet van toepassing zijn.

Deze verborgen subtiliteit was voor Deutsch de aanleiding tot de introductie van een fysische herformulering van de originele these. De bedoeling was om een ondubbelzinnig principe te formuleren dat vage begrippen, zoals 'functions which we would naturally regard as being computable,' adequaat specificeert. De nieuwe formulering heet het *principe van Church-Turing*.

> "Every finitely realizable system can be perfectly simulated by a universal model computing machine operating by finite means."
>
> David Deutsch, 1985

Dit principe omvat eenduidige begrippen die komen uit de moderne theorie van de fysica, kwantummechanica incluis. We beschrijven deze hieronder informeel.

finitely realizable system Dit zijn fysische objecten waarop men experimenten kan uitvoeren.

perfect simulation Het *perfect simuleren* onderscheidt zich van het *simuleren* uit de these van Church-Turing. Bij perfect simuleren worden geen discrete benaderingen geaccepteerd.

universal model computing machine Dit is een geïdealiseerd, eindig specificeerbaar berekeningsmodel. Een exacte overeenstemming met de werkelijkheid hoeft dus niet noodzakelijkerwijs te kunnen.

by finite means De eindigheid van het systeem wordt axiomatisch gedefinieerd: (i) tijdens een berekeningsstap is er steeds een eindig subsysteem in beweging, (ii) de beweging is alleen afhankelijk van het eindig subsysteem en (iii) de wiskundige regel die de beweging specificeert, is eindig.

Deutsch toont aan dat zijn principe sterker is dan de these van Church-Turing. De verklaring gebeurt als volgt. Beschouwen we bijvoorbeeld de continuïteit van de klassieke dynamica. De mogelijke toestanden van dergelijk fysisch systeem vormen een continuüm. Het systeem kan zich immers in oneindig veel toestanden bevinden. Een klassieke Turing machine kan dergelijk fysisch systeem niet perfect simuleren daar het een discreet berekeningsmodel is.

3.1.3 Introductie van universele kwantumcomputer

Omdat de klassieke fysica en de klassieke Turing niet voldoen aan het Church-Turing principe, ontwikkelde Deutsch een *universele kwantumcomputer*. Dit berekeningsmodel is wel in staat elk eindig fysisch realiseerbaar systeem perfect te simuleren en voldoet bijgevolg wel aan het principe van Church-Turing. De universele kwantumcomputer van Deutsch wordt in de literatuur vaak de *universele kwantum Turing machine* genoemd. De werking van deze machine wordt formeel uiteengezet in sectie 3.2.1.

Deutsch was onder andere beïnvloed door Benioff [Ben82] en Feynman [Fey82] die elk, op uitlopende manieren, een eerste aanzet tot de constructie van een universele kwantumcomputer hebben gesuggereerd.

Benioff ontwikkelde een berekeningsmodel dat kwantumtoestanden op de geheugenband van een Turing machine zette. De idee was om een Turing machine te ontwikkelen met omkeerbare berekeningen. Zijn inspiratie om met kwantumtoestanden te werken, is afkomstig van de unitaire evolutie van kwantummechanische systemen.

Feynman stelde zich als eerste de vraag of kwantummechanische systemen efficiënt kunnen gesimuleerd worden op een klassieke computer. Deze kwestie staat uiteengezet in sectie 4.2.2.

3.2 Kwantum Turing machine

3.2.1 Formele beschrijving

Een kwantum Turing machine (QTM) wordt gedefinieerd door het drietal $M = (Q, \Sigma, D)$. Q is een eindige verzameling met toestanden, die onder andere de begintoestand q_0 en de eindtoestand $q_f \neq q_0$ bevatten. De verzameling Σ is een eindig alfabet met een extra blanco symbool aangegeven door $\#$. De lokale transitiefunctie D karakteriseert de kwantum Turing machine, deze wordt verderop deze sectie toegelicht.

De QTM beschikt over een oneindig lange geheugenband met geïndexeerde cellen. De indices zijn aangegeven met de verzameling voor gehele getallen $\mathbb{Z}$. De verzameling van alle mogelijke geheugenband-configuraties wordt aangegeven met $\Sigma^{\mathbb{Z}}$. De geheugenband is voorzien van een lees- en schrijfkop die naar rechts $d = 1$ en naar links $d = -1$ kan bewegen.

Elke configuratie c van de machine M wordt beschreven als $c = (q, T, \xi) \in C$ met configuratieruimte $C = (Q \times \Sigma^{\mathbb{Z}} \times \mathbb{Z})$. Het eerste lid q geeft de huidige toestand weer, het tweede lid T geeft de inhoud van de geheugenband weer en het laatste lid ξ is de positie van de lees- en schrijfkop.

De toestand van de machine M wordt voorgesteld door een eenheidsvector $|\psi\rangle = |q\rangle \otimes |T\rangle \otimes |\xi\rangle$ in de Hilbertruimte, voortgebracht door de configuratieruimte C. Merk op dat de mogelijkheid op superposities tussen configuraties hier in rekenschap wordt gebracht $|\psi\rangle = \sum_i \alpha_i c_i$ met $\alpha_i \in \mathbb{C}$ en $c_i \in C$.

De berekeningen beginnen op tijdstip $t = 0$. De begintoestand van de machine M ziet er dan als volgt uit $|\psi(0)\rangle = |q_0\rangle|T_{in}\rangle|0\rangle$. De evolutie van de toestand van machine M wordt beschreven door een unitaire operator U uit de Hilbertruimte. Voor alle positieve gehele getallen t geldt dat:

$$|\psi(t)\rangle = U^t|\psi(0)\rangle \tag{3.1}$$

De functie $D(q, \sigma, q', \tau, d) = c$ met $c \in \mathbb{C}$ beschrijft een instructie voor de kwantum Turing machine. Vanuit toestand q haalt de leeskop het symbool σ op, we komen terecht in toestand q' met amplitude c. De schrijfkop schrijft symbool τ weg en verplaatst zich verder volgens d. De functie D is willekeurig onder de voorwaarde dat U een unitaire transformatie is. Elke verschillende keuze van D, geeft een andere machine M.

Tijdens de evolutie van de machine M kunnen de toestanden niet geraadpleegd worden zonder effect. Er dient dus een stop-schema geïntroduceerd te worden die aangeeft wanneer een berekening voltooid is. Dit wordt afgehandeld door $\widehat{n}_0 = |q_f\rangle\langle q_f|$. Het is de bedoeling dat na elke elementaire berekeningsstap, de $\widehat{n}_0$ gemeten wordt. Wanneer deze 1 is, stopt de machine.

3.2.2 Kwantumberekenbaarheid

Bennett [Ben73] toonde aan dat alle logische poorten uit de klassieke berekeningen een kwantumequivalent kennen.[1] Dergelijke omzetting kan efficiënt gebeuren. We kunnen hieruit concluderen dat een kwantumcomputer op zijn minst dezelfde klasse van functies kan berekenen als een klassieke computer. Om nu aan te tonen dat een kwantumcomputer dezelfde klasse van functies kan berekenen als een klassieke computer, illustreren we hoe een kwantumcomputer kan gesimuleerd worden op een klassieke Turing machine. Daar we enkel de berekenbaarheid nagaan, houden we geen rekening met de efficiëntie van de simulatie.

We encoderen de begintoestand van de kwantumcomputer in een kolommatrix van grootte n. Alle kwantumoperatoren worden vervat in een matrix met dimensie $n \times n$. Beide matrices bevatten uitsluitend complexe getallen. De kwantumberekening vertaalt zich dan in een matrixvermenigvuldiging.

Elk specifiek kenmerk van kwantumcomputers zal omgezet worden naar een equivalente klassieke berekening. Zo zal bijvoorbeeld *kwantumparallellisme* geserialiseerd worden in opeenvolgende lineaire bewerkingen of zal de *verstrengeling* impliciet beschreven zijn in de toestandsmatrix. Het werk van Bernstein en Vazinari [BV97] gaat na welke precisie vereist om het resultaat niet te schaden.

Een belangrijk detail is de afhandeling van de meting. Dit wordt gesimuleerd door willekeurige getallen te genereren. Door de kwantummechanische natuur dienen dit *werkelijke willekeurige getallen* te zijn die zich onderscheiden van *schijnbare willekeurige getallen* berekend door een deterministisch proces. Het simuleren van willekeurige getallen wordt gerealiseerd door de introductie van een muntworp-register in de Turing machine. Dergelijk speciaal alleen-lezen register kan geïmplementeerd worden met een niet-deterministische Turing machine. Deze laatste dient dan alle mogelijke berekeningspaden bij te houden van de muntworpen.

[1] In feite toonde hij aan dat alle klassieke logische circuits een circuit hebben waar omkeerbare berekeningen mogelijk zijn.

3.2.3 Kwantumcomplexiteit

Het berekeningsmodel van Turing gaf de aanleiding tot een nieuwe subtak in de wiskunde en informatica, namelijk de *complexiteitstheorie*. Het model van Turing kon niet alleen een antwoord formuleren op de vraag van wat berekenbaar is en wat niet, maar ook over hoe efficiënt de berekening verloopt. We gaan in deze sectie na hoe de klassieke complexiteitstheorie kan toegepast worden op kwantumcomputers.

Klassieke complexiteitsklassen

In deze sectie staat een beknopt overzicht van de klassieke complexiteitsklassen. Waar mogelijk werd de hiërarchie tussen de klassen weergegeven.

Klasse P Dit zijn de functies die in polynomische rekentijd kunnen berekend worden op een klassieke deterministische Turing machine. Deze klasse omvat problemen met orde van groei $O(n^c)$ met c een constante die machinegebonden is.

Klasse NP Dit is de niet-deterministische variant van de klasse P. Dit zijn de functies die in polynomische tijd kunnen berekend worden door een niet-deterministische Turing machine. Daar een niet-deterministische Turing machine simuleerbaar is op een deterministische Turing machine, geldt dat $P \subseteq NP$. Dit impliceert echter dat de uitvoeringstijd na simulatie exponentieel kan groeien. Een populaire vraag binnen de computerwetenschappen is of er problemen zijn in de klasse NP die niet in de klasse P zitten.

Klasse NP-compleet Dit zijn de zwaarste berekeningsproblemen binnen de klasse NP.

Klasse BPP Dit zijn problemen die berekend kunnen worden door een probabilistische Turing machine in polynomische rekentijd met maximale kans op fout $\frac{1}{3}$. Daar een probleem uit de klasse P kan gezien worden als een probabilistisch probleem met fout 0, geldt dat $P \subseteq BPP$. Open vragen zijn of $P = BPP$, $BPP \subseteq NP$ of $NP \subseteq BPP$.

Klasse PSPACE Dit zijn problemen die kunnen berekend worden door een Turing machine met een polynomische hoeveelheid bandgeheugen en oneindig veel berekeningstijd. De klasse EXPSPACE kent een equivalent definitie met uitzondering dat er een exponentiële hoeveelheid bandgeheugen nodig is. Men kan bewijzen dat $BPP \subseteq PSPACE$, $NP \subseteq PSPACE$ en $PSPACE \subseteq EXPSPACE$.

Kwantum complexiteitsklasse BQP

Het kwantumalternatief van de klasse BPP is BQP. Daarin zitten de problemen die in polynomische rekentijd kunnen berekend worden door een kwantum Turing machine met maximale kans op fout $\frac{1}{3}$. De hiërarchie $BPP \subseteq BQP$ gaat op daar elke klassieke berekening efficiënt kan gesimuleerd worden op een kwantumcomputer. Het is echter nog niet geweten of $BPP = BQP$.

Men kan bewijzen dat de klasse BQP zich binnen de complexiteitshiërarchie tussen de klasse BPP en $PSPACE$ bevindt. Dit betekent dat BQP alle problemen uit de klassen P en BPP bevat. Het verband met de klassen NP en NP-compleet is tot op heden onduidelijk. Er wordt gespeculeerd dat BQP vermoedelijk enkele problemen uit de klasse NP bevat maar niet uit de klasse NP-compleet.

Performantievoordeel van kwantumcomputers

De meest cruciale vraag is of kwantumcomputers 'algemeen efficiënter' zijn dan klassieke computers. Het antwoord is negatief. Alle problemen die efficiënt opgelost kunnen worden op een klassieke computer, kunnen ook efficiënt opgelost worden op een kwantumcomputer. De kwantumcomputer haalt enkel bij specifieke berekeningsproblemen een exponentieel performantievoordeel tegenover klassieke computers. Tot op heden is het onduidelijk over hoeveel dergelijke berekeningsproblemen het precies gaat.

Het exponentieel performantievoordeel van een kwantumcomputer zit hem in het fenomeen *kwantumparallellisme*. Wanneer een kwantumoperator wordt toegepast op een qubit, wordt deze in een enkelvoudige tijdseenheid toegepast op zijn 2^n superposities. Deze vorm van parallellisme kan echter maar in beperkte omstandigheden ingezet worden. Men kan immers de 2^n amplitudes van de superposities niet raadplegen als gevolg van de kwantummechanische natuur. Het vormt voor de wiskundigen en informatici een uitdaging algoritmes te formuleren die alsnog het performantievoordeel kunnen intensiveren. Een mooi voorbeeld hiervan is het algoritme van Shor [Sho96] die uitgebreid wordt uiteengezet in sectie 7.3.

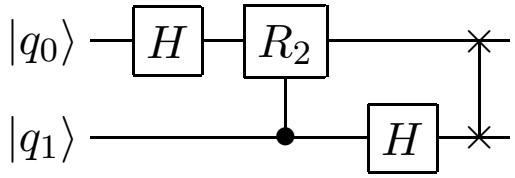
3.3 Kwantumcircuits

In voorgaande sectie introduceerden we de kwantum Turing machine. Dit berekeningsmodel ontleent zich uitstekend om de fundamenten van kwantumberekeningen, om bijvoorbeeld het begrip kwantumberekening formeel te kunnen vatten, in neer te schrijven. Net zoals bij de klassieke Turing machine,

is de kwantum Turing machine weining praktisch bruikbaar. In de praktijk worden hoofdzakelijk kwantumcircuits gebruikt om kwantumberekeningen in uit te drukken. Andrew Chi-Chih Yao [Yao93] heeft in 1993 aangetoond dat er voor elke kwantum Turing machine een equivalent kwantumcircuit bestaat. Dit kan efficiënt in polynomische rekentijd bepaald worden.

3.3.1 Bouwstenen van een kwantumcircuit

Als voorbeeld introduceren we het kwantumcircuit dat de discrete Fourier transformatie van twee enkelvoudige qubits $|q_0\rangle$ en $|q_1\rangle$ berekent. Vooraleer we alle particuliere componenten van een kwantumcircuit en de eigenschappen ervan in detail bespreken, geven we eerst een beknopte informele beschrijving van wat zich afspeelt in figuur 3.1.



Figuur 3.1: discrete Fourier transformatie op 2-qubit

We onderscheiden volgende stappen:

- Eerst wordt er een Hadamard operator toegepast op de qubit $|q_0\rangle$.
- In de volgende stap wordt de conditionele R_2 operator toegepast op de eerste qubit $|q_0\rangle$ met $|q_1\rangle$ als controle qubit.
- In de voorlaatste stap wordt de Hadamard operator toegepast op $|q_1\rangle$.
- Om af te sluiten worden beide qubits gewisseld.

Kwantumkanalen en meting

Een kwantumcircuit dient steeds van links naar rechts gelezen te worden. De horizontale lijnen stellen kwantumkanalen voor enkelvoudige qubits voor. Deze kanalen stemmen niet noodzakelijkerwijs overeen met fysieke kanalen. Ze suggereren gewoon de chronologische volgorde van gebeurtenissen of de locatie van een bepaald fysisch partikel doorheen de ruimte.

Wanneer een schuine streep is aangebracht op het kanaal, kan het meerdere qubits bevatten. Kanalen voor klassieke bits worden expliciet aangegeven

met een dubbele lijn. De meting wordt voorgesteld door middel van een gereserveerd symbool op het respectievelijke kwantumkanaal. Vorig benoemde symbolen staan weergegeven in tabel 3.1.

naam	symbool
enkelvoudige qubit	
meervoudige qubit	
klassieke bit	
meting	

Tabel 3.1: symbolen voor informatiekanalen en meting

Er zijn enkele restricties van toepassing voor kwantumcircuits die onnatuurlijk lijken voor klassieke circuits. We geven een opsomming en verklaren vanwaar deze restricties afkomstig zijn.

- Lussen zijn niet toegelaten in kwantumcircuits. Dit betekent dat alle kwantumcircuits *acyclisch* moeten zijn. Het gebruik van lussen zou berekeningen onomkeerbaar kunnen maken, wat in strijd is met de evolutie van kwantummechanische systemen (zie sectie 2.2.2).

- Twee of meerdere kwantumkanalen mogen niet samengebundeld worden. In klassieke circuits wordt dit wel toegestaan, het knooppunt gedraagt zich dan als een disjunctie. De disjunctie is niet omkeerbaar.

- Omgekeerd, een kwantumkanaal mag niet gesplitst worden naar meerdere verschillende kanalen als gevolg van de niet-klonen stelling.

Kwantumoperatoren

Een enkelvoudige kwantumoperator wordt voorgesteld met een rechthoek op het respectievelijke kwantumkanaal. In tabel 3.2 wordt een overzicht gegeven van enkele vaak voorkomende enkelvoudige kwantumoperatoren, samen met hun matrixvoorstelling.

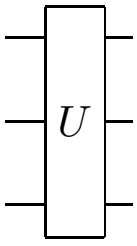
Meervoudige kwantumoperatoren die inwerken op verschillende aaneensluitende qubits, kunnen voorgesteld worden door rechthoeken die al deze qubits bestrijken. Een voorbeeld van dit laatste staat in figuur 3.2.

Conditionele kwantumpoorten

Een conditionele kwantumoperator is een (enkelvoudige of meervoudige) kwantumoperator vergezeld van één of meerdere controle qubits. Dergelijke type

naam	symbool	unitaire matrix
Hadamard	H	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Pauli-X	X	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli-Y	Y	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli-Z	Z	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Fase	S	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
$\pi/8$	T	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$

Tabel 3.2: vaak voorkomende enkelvoudige kwantumoperatoren

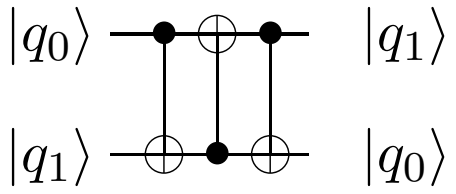


Figuur 3.2: voorstelling kwantumoperator U voor 3-qubit

operator wordt alleen toegepast wanneer alle conditionele qubits dit toestaan. Dit is het equivalent van een als...dan constructie binnen de klassieke berekeningen.

Conditionele operatoren worden gevisualiseerd als gewone operatoren, maar zijn voorzien van verlengstukken naar de respectievelijke conditionele kanalen samen met een dikke bol. In tabel 3.3 staan enkele vaak voorkomende conditionele kwantumoperatoren weergegeven.

Speciale aandacht gaat uit naar de wisseloperator. Zijn functie bestaat erin om twee particuliere qubits van plaats te verwisselen, bijvoorbeeld $|q_0 q_1\rangle \rightarrow |q_1 q_0\rangle$. Dit is in feite een impliciete conditionele kwantumoperator daar deze geïmplementeerd kan worden door middel van uitsluitend conditionele negatie-operatoren. Dit staat geïllustreerd in figuur 3.3.



Figuur 3.3: implementatie van de wissel-operator

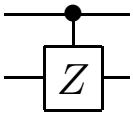
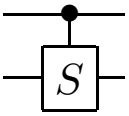
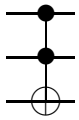
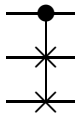
We kunnen deze implementatie als volgt algebraïsch verklaren.

$$\begin{aligned} |a,b\rangle &\rightarrow |a, a \oplus b\rangle \\ &\rightarrow |a \oplus (a \oplus b), a \oplus b\rangle = |b, a \oplus b\rangle \\ &\rightarrow |b, (a \oplus b) \oplus b\rangle = |b, a\rangle \end{aligned} \tag{3.2}$$

3.3.2 Universele kwantumoperatoren

Binnen de klassieke berekeningen volstaan de operatoren conjunctie, disjunctie en negatie om een willekeurige berekening in uit te drukken. Men noemt deze discrete verzameling van operatoren *de universele operatoren*. We gaan in deze sectie na of er een discrete verzameling van kwantumoperatoren bestaat waarmee elk willekeurig kwantumcircuit kan uitgedrukt worden.

De moeilijkheid bij het specificeren van een discrete verzameling van universele kwantumoperatoren ligt hem in het feit dat de kwantummechanische natuur continu is. Strikt gezien is het niet mogelijk om met een eindig aantal elementen, een continue wereld te beschrijven. We gaan dus op zoek naar kwantumoperatoren die alle andere operatoren voldoende nauwkeurig

naam	symbool	unitaire matrix
conditionele negatie		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
wissel		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
conditionele Z		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$
conditionele fase		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{pmatrix}$
Toffoli		$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$
Fredkin		$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$

Tabel 3.3: vaak voorkomende conditionele kwantumoperatoren

kunnen benaderen. Men kan bewijzen dat de Hadamard (H), fase (S), conditionele negatie ($CNOT$) en $\pi/8$ operatoren aan deze voorwaarde voldoen. We noemen ze de universele kwantumoperatoren.

Een belangrijke opmerking is dat, gegeven een willekeurige kwantumoperator U die inwerkt op n qubits, er niet altijd een kwantumcircuit bestaat met grootte een polynoom van n. Op basis van de stelling van Solovay-Kitaev [Kit97] kan men aantonen dat, gegeven een fout ϵ, het aantal benodigde universele kwantumoperatoren $\Omega(n^2 4^n \log_c(n^2 4^n/\epsilon))$ bedraagt. Informeel betekent dit dat kwantumoperatoren niet altijd efficiënt benaderd kunnen worden.

3.3.3 Voorbeeld: kwantum teleportatie

We illustreren het gebruik van kwantumcircuits om het begrip kwantum teleportatie te verklaren. Vooraleer we het kwantumcircuit bespreken, lichten we eerst informeel toe wat kwantum teleportatie precies betekent, hoe dit mogelijk wordt gemaakt en waarom dit precies zo belangrijk is. Merk op dat het kwantum teleportatie voorbeeld in sectie 6.1.2 wordt hervat om uit te leggen hoe kwantumcircuits kunnen berekend worden.

Kwantum teleportatie is een techniek om kwantumtoestanden te versturen over een zekere afstand zonder dat daarbij een kwantum communicatiekanaal bij te pas komt. Als gevolg van de kwantummechanische natuur, mag de te versturen qubit niet gemeten worden. Anders zou de qubit vervallen in één van zijn discrete meetwaardes. Daarnaast is het strikt genomen niet mogelijk om via een klassiek communicatiekanaal een qubit te versturen daar deze zich in oneindig veel toestanden kan bevinden.

Door gebruik te maken van de merkwaardige eigenschappen van een verstrengeld EPR-paar, zoals beschreven in sectie 2.2.4, kan de communicatie toch tot stand gebracht worden. Beschouwen we een zender (Alice) en een ontvanger (Bob) die samen een EPR-paar $|\beta_{00}\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$ genereren. Alice en Bob nemen elk een qubit van het verstrengelde EPR-paar en daarna splitsen ze zich.

De bedoeling is nu dat Alice een willekeurige qubit $|\psi\rangle = a|0\rangle + b|1\rangle$ verstuurt naar Bob. Daarom doet Alice een beroep op haar qubit van het verstrengelde EPR-paar $|\beta_{00}\rangle$. Dit gebeurt als volgt.

- Alice laat haar qubit $|\psi\rangle$ interageren met haar helft van het EPR-paar $|\beta_{00}\rangle$.
- Vervolgens meet Alice haar twee qubits. De meting kan resulteren in één van de volgende klassieke uitkomsten 00, 01, 10 of 11.

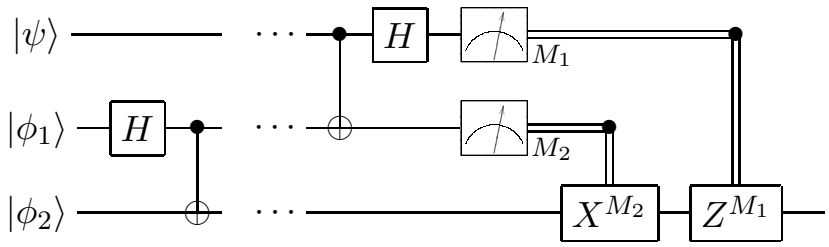
- Alice stuurt via een klassiek communicatiekanaal haar meetresultaat m naar Bob.
- Op basis van het meetresultaat m kan Bob de juiste decodeer-operator uit tabel 3.4 kiezen om de qubit $|\psi\rangle$ te reconstrueren uit zijn helft van het EPR-paar $|\beta_{00}\rangle$.

Een belangrijke opmerking is of de mogelijkheid bestaat om informatie sneller dan het licht te versturen met behulp van kwantum teleportatie. De relativiteitstheorie zegt dat het versturen van informatie sneller dan het licht, de mogelijkheid biedt om informatie naar het verleden te sturen. Jammergenoeg is communicatie sneller dan het licht niet mogelijk omdat Alice via een klassiek communicatiekanaal haar meetresultaat m moet doorgeven aan Bob.

ontvangen bits	decodeer-operator
00	I
01	X
10	Z
11	ZX

Tabel 3.4: decodeer operatoren

Het kwantumcircuit voor teleportatie ziet er als volgt uit.



Figuur 3.4: kwantum teleportatie

We beschrijven het verloop van kwantum teleportatie algebraïsch op basis van een willekeurige qubit $|\psi\rangle = a|0\rangle + b|1\rangle$ die door Alice werd geconstrueerd. De begintoestand $|\psi_0\rangle$ van het kwantumcircuit ziet er als volgt uit.

$$\begin{aligned} |\psi_0\rangle &= |\psi\rangle \otimes |\phi_1\rangle \otimes |\phi_2\rangle \\ &= |\psi, 0, 0\rangle \end{aligned} \tag{3.3}$$

Terwijl Alice en Bob nog samen zijn, genereren ze het EPR-paar $|\beta_{00}\rangle = |\phi_1, \phi_2\rangle$. Nadien nemen Alice en Bob elk een qubit, respectievelijk $|\phi_1\rangle$ en $|\phi_2\rangle$, van het verstrengelde EPR paar. Het feit dat Alice en Bob zich vanaf nu splitsen, wordt in figuur (3.4) weergegeven door stippellijntjes. De kwantumtoestand ziet er nu als volgt uit.

$$\begin{aligned} |\psi_1\rangle &= |\psi\rangle \otimes |\beta_{00}\rangle \\ &= \frac{1}{\sqrt{2}}\left[a|0\rangle(|00\rangle + |11\rangle) + b|1\rangle(|00\rangle + |11\rangle)\right] \end{aligned} \tag{3.4}$$

Alice past nu de conditionele negatie toe op haar eerste qubit $|\phi_1\rangle$ van het EPR-paar $|\beta_{00}\rangle$ met de qubit $|\psi\rangle$ als conditie. Dit geeft ons volgende toestand.

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}\left[a|0\rangle(|00\rangle + |11\rangle) + b|1\rangle(|10\rangle + |01\rangle)\right] \tag{3.5}$$

Daarna past Alice de Hadamard kwantumoperator toe op haar qubit $|\psi\rangle$. We bekomen de volgende toestand.

$$\begin{aligned} |\psi_3\rangle &= \frac{1}{2}\left[a(|0\rangle + |1\rangle)(|00\rangle + |11\rangle) + b(|0\rangle - |1\rangle)(|10\rangle + |01\rangle)\right] \\ &= \frac{1}{2}[|00\rangle(a|0\rangle + b|1\rangle) + |01\rangle(a|1\rangle + b|0\rangle) \\ &\quad + |10\rangle(a|0\rangle - b|1\rangle) + |11\rangle(a|1\rangle - b|0\rangle)] \end{aligned} \tag{3.6}$$

Alice voert nu een partiële meting uit op haar qubits $|\psi\rangle$ en $|\phi_1\rangle$. De kans op de mogelijke meetresultaten $m = 00$, 01, 10 of 11 bedraagt steeds $p(m) = 1/4$. We krijgen bijgevolg vier mogelijke toestanden $|\psi_4(m)\rangle$ na meting.

$$00 \longmapsto |\psi_4(00)\rangle = a|0\rangle + b|1\rangle \tag{3.7}$$

$$01 \longmapsto |\psi_4(01)\rangle = a|1\rangle + b|0\rangle \tag{3.8}$$

$$10 \longmapsto |\psi_4(10)\rangle = a|0\rangle - b|1\rangle \tag{3.9}$$

$$11 \longmapsto |\psi_4(11)\rangle = a|1\rangle - b|0\rangle \tag{3.10}$$

Bob kan nu de oorspronkelijke toestand van $|\psi\rangle$ reconstrueren door de juiste kwantumoperator(en) toe te passen op $|\psi_4(m)\rangle$ zoals beschreven in tabel 3.4.

Hoofdstuk 4

Simulatie van kwantumberekeningen

Aan de hand van zeven objectieven preciseren we binnen welke filosofie de kwantumsimulator is ontwikkeld. De bedoeling is om een bestaande klassieke programmeertaal uit te breiden met abstracte datatypes om kwantumberekeningen in uit te drukken. Het resultaat is een expressieve, korte en efficiënte multiparadigma taaluitbreiding in Lisp. We motiveren de deelname aan het wetenschappelijk debat door aan te tonen hoe de objectieven soms haaks staan met reeds bestaande kwantumsimulatoren.

4.1 Introductie

Dit hoofdstuk werpt de schijnwerpers op alle mogelijke aspecten die bij de constructie van een kwantumsimulator komen kijken. Doorheen de uiteenzetting van de verschillende kwesties, leiden we zeven objectieven af. Deze beschrijven de ontwikkelingsfilosofie van de simulator en fungeren als motivatiebodem voor diverse ontwerpbeslissingen. In de hoofdstukken 5 en 6, waar de implementatie van de simulator wordt behandeld, wordt veelvuldig teruggeblikt op deze objectieven.

4.2 Situering van de kwantumsimulator

4.2.1 Behoefte aan simulatie

We hernemen dat een kwantumberekening wordt gespecificeerd in een berekeningsmodel. Het meest gebruikte model voor praktische doeleinden zijn *kwantumcircuits* zoals beschreven in sectie 3.3. Dergelijke circuits kunnen *berekend* worden met behulp van een *kwantumcomputer*. Dit is een computer die opgebouwd is uit atomaire partikels waarvan de evolutie wordt bepaald door de postulaten van de kwantummechanica.

Gebrek aan hardwarematige implementaties

Het onderzoek rond de ontwikkeling van hardwarematige realisaties is de laatste jaren sterk toegenomen. Intussen zijn er al verschillende succesvolle implementaties beschikbaar van kwantumcircuits, bijvoorbeeld voor het probleem van Deutsch dat in sectie 7.1 wordt behandeld (Jones & Mosca, 1998). Er bestaat echter, tot op heden, nog geen algemene kwantumcomputer. Verdere ontwikkelingen hangen af van de vooruitgang binnen de nanotechnologie. Om toch het gebied van kwantumcomputers verder te kunnen verkennen, zonder een beroep te kunnen doen op een werkelijke implementatie, dringt het gebruik van een simulator zich op.

Overstijging van het derde postulaat

Specifiek voor onderzoeksdoeleinden, brengt een kwantumsimulator extra voordelen met zich mee. Een simulator biedt de mogelijkheid om steeds de interne toestand van een kwantumsysteem te observeren, zonder het systeem te verstoren. Dit laatste is niet realiseerbaar op een kwantumcomputer als gevolg van het derde postulaat van de kwantummechanica. Via de simulator kunnen we dus op een gedetailleerde manier de mogelijkheden van kwantumalgoritmes gaan verkennen. Dit stimuleert het onderzoek en de ontwikkeling van nieuwe kwantumalgoritmes.

Onderzoek rond kwantum foutcorrectie

Een kwantumsimulator is ook een handig instrument om de karakteristieken van kwantum foutcorrectie, en de accumulatie daarvan, te onderzoeken (Obenland & Despain, 1997). De effectiviteit van kwantumberekeningen kan immers verstoord worden door de aanwezigheid van fouten. Simulatoren maken het mogelijk om de oorzaak en het gevolg van verschillende soorten fouten te bestuderen. Een programmeertaal zoals Lisp biedt ons bijvoorbeeld

de mogelijkheid de simulatie te onderbreken, het gedrag van de simulator te manipuleren en de berekeningen verder te zetten.

Samengevat

We vatten de behoeftes uit deze sectie samen in onderstaand eerste objectief voor de constructie van een eigen simulator.

Objectief 1 *Het ontwikkelen van een simulator met flexibele experimenteermogelijkheden om onderzoek en ontwikkeling te stimuleren.*

4.2.2 Simuleren van fysica met computers

Een *kwantumsimulator* is een computerprogramma dat de acties, gespecificeerd door een kwantum berekeningsmodel, uitvoert op een klassieke computer. Deze computer is gebaseerd op de klassieke wetten van de Newtoniaanse fysica en simuleert de wetten van de kwantummechanica. De beroemde fysicus Feynman stelde zich in 1982 voor het eerst de vraag of het mogelijk is om de wetten van de kwantummechanica te simuleren met computers [Fey82]. Zijn mening luidt als volgt.

> "[...] the full description of quantum mechanics for a large system with R particles [...] has too many variables, it cannot be simulated with a normal computer [...]"

Vrij vertaald beweert Feynman dat een kwantummechanisch systeem, bijvoorbeeld een R-qubit, niet kan gesimuleerd worden op een normale computer. We bekijken nader wat hij daarmee precies bedoelt. Feynman's idee van *simuleren* kan als volgt begrepen worden. Er dient een proportionele verhouding te bestaan tussen het aantal benodigde computerelementen en de grootte van het fysisch systeem. Een *normale computer* is een willekeurig berekeningsmodel dat kan uitgevoerd worden door een klassieke universele Turing machine.

Feynman motiveert zijn standpunt door twee verschillende (klassieke) computermodellen te bespreken om de probabilistische natuur van de kwantummechanica te simuleren. Hij toont aan dat beide manieren niet volstaan om over werkelijke simulaties te kunnen spreken. We geven een beknopte samenvatting van zijn mening.

Rekenapparaat voor probabiliteiten

In een eerste poging om de probabilistische natuur te simuleren, gebruiken we *de computer als een rekenapparaat voor probabiliteiten*. Het geheugen van een computer is eindig. De voorstelling van de probabiliteiten zal dus gediscretiseerd moeten worden tot op bijvoorbeeld k cijfers. Een probabiliteit kleiner dan 2^{-k} zal bijgevolg als onmogelijk beschouwd worden door de computer, hoewel het in de natuur wel mogelijk kan zijn. Het tweede argument die een simulatie onmogelijk maakt, is de exponentiële complexiteit van het kwantummechanisch systeem. Immers, een R-qubit wordt voorgesteld door middel van 2^R amplitudes. Feynman noemt deze laatste "to many variables".

Probabilistische computer

Een andere mogelijkheid is om de probabilistische natuur te simuleren met een *probabilistische computer*. Dit zou een computer kunnen zijn die bijvoorbeeld steeds de laatste k cijfers van een getal vervangt met willekeurige cijfers. Op die manier is het resultaat geen unieke functie van de invoer. Als we nu een initiële toestand a laten evolueren naar een eindtoestand b met behulp van een probabilistische computer, zal de natuur niet noodzakelijkerwijs hetzelfde resultaat b geven op basis van dezelfde begintoestand a. Feynman beweert daarenboven dat de natuur en de probabilistische computer dit zelfs niet doen met dezelfde probabiliteit. Dit komt omdat de natuur onvoorspelbaar is, en dus ook niet voorspelbaar met een probabilistische computer. Daarnaast verwijst hij opnieuw naar de exponentiële complexiteit om bepaalde kwantummechanische effecten (superpositie of verstrengeling) te beschrijven.

4.2.3 Problemen met kwantumsimulatoren

We hernemen stap voor stap de problematiek die Feynman heeft aangekaart om een kwantumsimulator te bouwen. Voor elk probleem bekijken we welke impact dit heeft op de constructie van een eigen simulator. Het zal blijken dat de definitie van simuleren, zoals beschreven door Feynman, noodgedwongen vereenvoudigd zal moeten worden. Deze vereenvoudiging beschrijven we door middel van drie objectieven. In sectie 4.3 worden nog enkele objectieven toegevoegd die meer te maken hebben met aspecten uit de programmeerkunde.

Discretisering

Het is strikt genomen onmogelijk om, gegeven een eindig geheugen, de toestand van een qubit te beschrijven. Immers, een qubit kan zich in oneindig veel toestanden bevinden. Dit betekent dat een exacte simulatie niet realiseerbaar is. Bernstein en Vazinari [BV97] toonden aan dat er een ondergrens kan gedefinieerd worden voor de nauwkeurigheid van de discretisatie, zonder de kwantumberekeningen te schaden. Merk op dat er voortdurend een inherente fout aanwezig zal zijn in de simulator.

Deze benaderingsfout is voor onze doelstellingen eigenlijk niet erg. De behoefte is om kwantumalgoritmes te kunnen verkennen en ontwikkelen op een manier die niet realiseerbaar is op een werkelijke kwantumcomputer. Een eventuele benaderingsfout zal het resultaat van de kwantumberekening niet schaden. We formuleren dit probleem als een tweede objectief voor de ontwikkeling van een eigen simulator.

Objectief 2 *Het kunnen simuleren van willekeurige kwantumalgoritmes met inbegrip van een zekere verwaarloosbare fout als gevolg van discretisatie.*

Ruimte en tijd

Het meest fundamentele probleem dat Feynman aanhaalde, is de *exponentiële complexiteit* van de kwantummechanische natuur. Concreet betekent dit dat een n-qubit wordt voorgesteld als een eenheidsvector door een n-dimensionale Hilbertruimte. Deze eenheidsvector wordt beschreven met 2^n complexe getallen. De voorstelling van kwantuminformatie door middel van klassieke informatie kent dus een exponentiële ruimtecomplexiteit.

Hetzelfde verhaal speelt zich af bij de kwantumberekeningen. Wanneer een kwantumoperator wordt toegepast op een n-qubit, wordt deze in een enkele tijdseenheid toegepast op de 2^n superpositietermen. Dit fenomeen, kwantumparallellisme, dient sequentieel berekend te worden met een klassieke computer. Het introduceren van bijkomende verwerkingseenheden kan de rekentijd enkel met een linaire factor verminderen. De tijdscomplexiteit blijft inherent exponentieel.

Het is tot op heden niet mogelijk om een efficiënte implementatie van een kwantumsimulator te ontwikkelen. Immers, in dat geval zou het in principe niet meer interessant zijn om kwantumcomputers hardwarematig te implementeren. We concluderen met onderstaand objectief inzake efficiëntie.

Objectief 3 *Het gebruik van algemene optimalisatietechnieken die snoeien in de exponentiële groei zowel in rekentijd als in geheugengebruik.*

Probabilistische natuur

Een laatste struikelblok, wat Feynman de "onvoorspelbaarheid van de natuur" noemt, is de simulatie van de meting. Dit kwantummechanisch concept, beschreven in sectie 2.2.3, kan gesimuleerd worden met behulp van willekeurige getallen. Het is niet mogelijk om met een (deterministische) klassieke computer, werkelijke willekeurige getallen te berekenen. Dit betekent dat een *perfecte muntworp*, beschreven door de meting van de qubit $|\psi\rangle = 1/\sqrt{2}|0\rangle + 1/\sqrt{2}|1\rangle$ waarbij $p(0) = 1/2$ en $p(1) = 1/2$, onmogelijk waarheidsgetrouw kan gesimuleerd worden. In sectie 3.2.2 is aangetoond dat deze beperking de berekenbaarheid van de gesimuleerde kwantumberekeningen niet beschadigt. Daarom kunnen we een beroep doen op algoritmes die schijnbare willekeurige getallen genereren met een lage correlatie. In simulaties waar het probabilistische gedrag heel kritisch is, kan additionale hardware gebruikt worden in combinatie met een klassieke computer om werkelijke willekeurige getallen te genereren. Dit geeft ons volgend objectief.

Objectief 4 *De simulatie van de probabilistische natuur van de kwantummechanica is slechts een deterministische benadering van de werkelijkheid.*

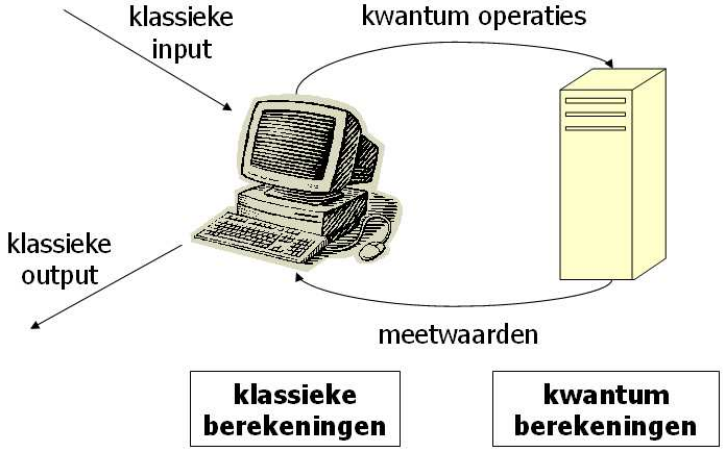
4.3 Kwantum programmeeromgeving

4.3.1 Motivatie voor hybride architectuur

Het performantievoordeel van een kwantumcomputer is een gevolg van kwantumparallellisme. Wanneer een operator wordt toegepast op een n-qubit, wordt deze operator in een enkele tijdseenheid toegepast op al zijn 2^n superpositietermen. In tegenstelling tot een parallelle computer, is het bij een kwantumcomputer niet mogelijk om de waarde van de amplitudes zonder effect te raadplegen. Voor bepaalde berekeningsproblemen (zie hoofdstuk 6) zal een kwantumcomputer effectief een exponentieel performantievoordeel halen in vergelijking met een klassieke computer. Het berekenen van een gewone optelsom gebeurt daarentegen even efficiënt op een kwantumcomputer als op een klassieke computer.

Het is dus niet altijd noodzakelijk om een beroep te doen op een kwantumcomputer. Alles wat efficiënt kan gebeuren op een klassieke computer, dient uitgevoerd te worden op een klassieke computer. Enkel specifieke deelproblemen die niet efficiënt op een klassieke computer kunnen uitgevoerd worden, moeten afgehandeld worden door een kwantumcomputer. Dit verklaart waarom er werd gekozen om *een bestaande klassieke programmeertaal uit te breiden met een simulator voor kwantumberekeningen.* We gebruiken

daarbij het *state-of-the-art kwantumcircuit berekeningsmodel* om kwantumberekeningen uit te drukken. Het heterogene karakter van de combinatie klassiek en kwantum resulteert in een *hybride architectuur*. Deze is grafisch voorgesteld in figuur 4.1.



Figuur 4.1: hybride architectuur

4.3.2 Motivatie voor integratie

In werkelijkheid is de kwantumprocessor en het kwantumgeheugen fysisch volledig gescheiden van het klassieke gedeelte. Een belangrijke ontwerpbeslissing is of deze fysische scheiding weerspiegelt moet worden in een programmeertaal. We motiveren de keuze om deze scheiding op het niveau van een programmeertaal te minimaliseren. De conclusie is dat een integratie van klassieke en kwantum programmeerconcepten het beste aansluit bij onze behoeftes om een kwantumsimulator te ontwikkelen.

IJzeren gordijn tussen klassiek en kwantum

Een scheiding tussen kwantum en klassiek in een programmeertaal, dit is het zogenaamde "ijzeren gordijn", zou concreet kunnen betekenen dat er een *kwantumprocessor*-module[1] bestaat. De belangrijkste parameter van deze module zou de beschrijving van een kwantumcircuit zijn. De taak van de module bestaat erin om de kwantumberekeningen uit te voeren die gespecificeerd zijn door het circuit. Het eindresultaat van de module zou onder

[1]Een module is een abstractiemechanisme voor een bepaald programmeerparadigma. In Lisp zou dit bijvoorbeeld een functie kunnen zijn.

andere de gevraagde meetresultaten moeten bevatten. De voordelen van deze methode staan in onderstaande opsomming.

- We verkijgen een heel duidelijke structuur: er is geen verwarring tussen klassieke en kwantum concepten mogelijk.
- Doordat de volledige kwantumberekening op voorhand bekend is, kunnen er optimalisatietechnieken toegepast worden die de simulatie efficiënter maken. Dit idee is vergelijkbaar met de optimalisatietechnieken die in compilers zitten.

De nadelen zijn:

- De taal om kwantumcircuits in uit te drukken zal waarschijnlijk een domeinspecifieke taal (afgekort met DSL) zijn. Dit betekent dat je onderhevig bent aan de *flexibiliteit* van deze DSL om kwantumcircuits in uit te drukken.
- Bepaalde *kwantumcircuits zijn dynamisch*, dat wil zeggen dat de grootte en/of het aantal gebruikte kwantumoperatoren in functie staat van een specifiek berekeningsprobleem. In sommige gevallen kan een intermediare stap nodig zijn om een instantie van een kwantumcircuit te genereren.
- Het *interactieve karakter* dat nodig is voor onderzoek en ontwikkeling van kwantumalgoritmes raakt verloren. Dit komt omdat het opvragen van bijvoorbeeld tussenresultaten van kwantumberekeningen omslachtiger zal worden.

Integratie van klassiek en kwantum

De beperktheid van het interactieve karakter (derde nadeel) doet de keuze voor een volledige scheiding tussen klassiek en kwantum de das om. De experimenteermogelijkheden behoren immers tot één van de elementaire behoeftes van de kwantumsimulator zoals beschreven door het eerste objectief. Daarom kiezen we voor het tegengestelde van een scheiding tussen klassiek en kwantum. We spreken over een *integratie van klassiek en kwantum*. Dit betekent dat klassieke en kwantum concepten door elkaar kunnen gebruikt worden én dat ze elkaar wederzijds aanvullen.

De voordelen van een integratie van klassiek en kwantum stemmen overeen met de nadelen van een scheiding. Het tegenovergestelde geldt voor de nadelen, we stellen volgende oplossingen voor.

- Het onderscheid tussen klassieke en kwantumconcepten wordt geaccentueerd door middel van een *code conventie*. Deze wordt geïntroduceerd in hoofdstukken 4 en 5.

- Een willekeurige kwantumberekening kan steeds berekend worden met behulp van een *algemeen simulatie-algoritme*. Bepaalde specifieke kwantumberekeningen kunnen geoptimaliseerd worden. Deze *particuliere simulatie-algoritmes* zijn ondergebracht in aparte functies. De behandelde simulatie-algoritmes uit hoofdstuk 6 kunnen dus steeds als "algemeen" of "specifiek" geklassificeerd worden.

4.3.3 Kwantum abstracte data types

We beschrijven kwantumcircuits met behulp van kwantum abstracte data types (afgekort ADT). We onderscheiden volgende kwantum ADT's.

- **Kwantumregister** Dit is het equivalent van een variabele uit een klassieke programmeertaal. Een kwantumregister is een abstractie van een willekeurige n-qubit. De implementatie van deze component wordt behandeld in hoofdstuk 5.

- **Kwantumoperator** Zoals een klassieke operator inwerkt op variabelen, werkt een kwantumoperator in op kwantumregisters. We onderscheiden twee soorten.

  - **gewone operatoren**
  - **conditionele operatoren** Deze worden slechts toegepast wanneer een conditioneel kwantumregister gezet is.

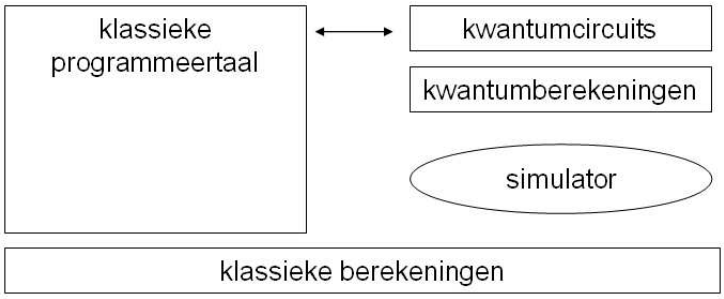
- **Meting** De observatie van een kwantumregister gebeurt door middel van een meting. We onderscheiden twee soorten.

  - **globale meting** Een volledig kwantumregister wordt gemeten.
  - **partiële meting** Slechts een deelverzameling qubits van een kwantumregister wordt gemeten.

  De implementatie van kwantumoperatoren en meting worden behandeld in hoofdstuk 6.

Deze kwantum ADT's zijn schijnbaar "gelijkwaardig" met klassieke ADT's, maar intern doen kwantum ADT's een beroep op de kwantumsimulator. De taak van de simulator is om kwantumberekeningen om te zetten in klassieke

berekeningen. De rechterhelft van figuur 4.2 visualiseert deze werkwijze. De linkerhelft van deze figuur visualiseert de klassieke werkwijze. De pijl tussen "klassieke programmeertaal" en "kwantumcircuit" verwijst naar de integratie van klassiek en kwantum.



Figuur 4.2: integratie van klassiek en kwantum

We vatten de sectie 4.3.1, 4.3.2 en 4.3.3 samen in volgend objectief.

Objectief 5 *Een bestaande klassieke programmeertaal uitbreiden met de nodige abstracte data types om kwantumberekeningen in uit drukken. De heterogene structuur tussen klassiek en kwantum resulteert in een hybride architectuur. Een integratie laat toe klassieke en kwantum concepten door elkaar te gebruiken en elkaar wederzijds aan te vullen waar nodig.*

Daarnaast wensen we over compacte abstracties te beschikken met een hoge uitdrukkingskracht. In hoofdstukken 5 en 6 illusteren we nader wat we met dit objectief precies bedoelen aan de hand van voor- en tegenvoorbeelden.

Objectief 6 *Het ontwikkelen van abstractiemechanismes die resulteren in een hoge uitdrukkingskracht voor kwantumberekeningen.*

4.3.4 Abstractiebarrière tussen klassiek en kwantum

We leggen de klemtoon op het simulatiegedeelte, dat eigenlijk een abstractielaag is tussen de kwantum en klassieke berekeningen. Het formalisme achter kwantumberekeningen is de lineaire algebra. Zo is een n-qubit eigenlijk een eenheidsvector met 2^n complexe getallen in een complexe vectorruimte $\mathbb{C}^n$. Een kwantumoperator die inwerkt op een n-qubit is eigenlijk een vierkante matrix met dimensie $2^n \times 2^n$. Het combineren van enkelvoudige qubits tot meervoudige qubits houdt een uitbreiding van de complexe vectorruimte in, dit wordt gerealiseerd door middel van een tensorproduct. Een gedetailleerde

uiteenzetting van het volledige formalisme (de postulaten van de kwantummechanica) staat beschreven in hoofdstuk 2.

De link tussen de kwantum concepten en de lineaire algebra is in de implementatie geconcretiseerd door middel van twee grote abstractielagen. Een eerste klassieke abstractielaag is een soort van *wiskundige bibliotheek* die data en operaties uit de lineaire algebra bevat. Daar bovenop is een tweede kwantum abstractielaag gecreëerd die *kwantumconcepten* doorgeeft aan de concepten uit de lineaire algebra. Dit idee zit vervat in onderstaand objectief.

Objectief 7 *Eenduidige abstractiebarrière voor de omzetting van kwantumberekeningen in klassieke berekeningen.*

4.4 Kwantumsimulator in Lisp

De eerste Lisp ideeën werden ontwikkeld door John McCarthy in 1958, met het oog op het ontwikkelen van een programmeertaal voor artificiële intelligentie. Dit werk werd semi-onafhankelijk door verschillende instanties verder uitgewerkt tot begin de jaren '80. Op die manier raakten vele Lisp-dialecten in circulatie. Guy Steele [Ste84] beschreef in 1984 een gestandaardiseerde versie van Lisp met de naam Common Lisp. De tweede editie [Ste89] van zijn werk verscheen 5 jaar later en fungeerde als de facto standaard voor veel Lisp-gebruikers gedurende enkele jaren.

In 1986 werd een technische werkgroep X3J13 opgericht die de specificaties van ANSI Common Lisp ontwikkelde. Enkele doelstellingen van deze nieuwe standaard waren een betere platformonafhankelijkheid, de integratie van het object-georiënteerd programmeerparadigma en een uitbreiding van de iteratiefaciliteiten. Deze standaard werd gepubliceerd in 1995. De stabiliteit is bewezen doordat ANSI Common Lisp tot op heden nog steeds in gebruik is. Voor een meer gedetailleerde uiteenzetting van de geschiedenis van Lisp, verwijzen we naar [McC78] en [SG96].

Op Fortran na, is Lisp de oudste programmeertaal die nog steeds in gebruik is. Het is merkwaardig hoe deze programmeertaal haar plaats vertegenwoordigt in de frontlinie van de moderne programmeertechnologie. De redenen hiervoor fungeren als motivatie om de kwantumsimulator te ontwikkelen in ANSI Common Lisp.

4.4.1 Taaluitbreiding in Lisp

De essentie van onze kwantumsimulator is een taaluitbreiding zoals beschreven door het vijfde objectief, "een bestaande klassieke programmeertaal uit-

breiden [...]". We tonen in deze sectie nauwgezet aan waarom Lisp geschikt is om deze opdracht te vervullen. De discussie is gebaseerd op een opinietekst van Paul Graham uit zijn boek "ANSI Common Lisp" [Gra95]. De kern van het hele verhaal zit vervat in het volgende citaat.

> "Programming languages teach you not to want what they cannot provide."
>
> Paul Graham, programmeertaal designer

De betekenis van dit citaat kan men als volgt begrijpen. Elke programmeertaal beschikt over een aantal ingebouwde abstractietechnieken. Wanneer je een berekeningsprobleem wil oplossen in een bepaalde programmeertaal, beschrijf je dat probleem op basis van de beschikbare abstractietechnieken. We noemen deze aanpak *van boven naar beneden* programmeren. De implementatie kan sterk bemoeilijkt worden wanneer de programmeertaal tekort schiet in passende abstractietechnieken voor het probleem in kwestie. We illustreren dit idee met een eenvoudig voorbeeld. Wanneer een programmeur bijvoorbeeld in BASIC schrijft, wordt hij impliciet aangemoedigd om te werken met iteratieve algoritmes. Dit komt omdat deze programmeertaal doorgaans geen recursie ondersteund. Er bestaat ook geen mogelijkheid om een taaluitbreiding te definiëren die recursie wel mogelijk zou maken.

Dit laatste is de hoofdoorzaak waarom een programmeertaal kan uitgespeeld raken na verloop van tijd. Je bent inherent gelimiteerd aan het paradigma en de abstractietechnieken van de programmeertaal in kwestie. Lisp vormt daarop een uitzondering. Het is immers mogelijk om zelf nieuwe operatoren te definiëren naast de standaard ingebouwde operatoren. Dit komt omdat Lisp gedefinieerd is van dezelfde functies en macro's die we gebruiken om te programmeren in Lisp. De eenvoud van dit concept zorgt ervoor dat een *taaluitbreiding*, de standaard manier van werken is in Lisp. In plaats van software alleen *van boven naar beneden* te schrijven, zoals in het voorbeeld van BASIC, schrijven we ook *van beneden naar boven* [AS87]. We verklaren nader wat dit precies betekent.

De bedoeling van een taaluitbreiding is om *abstractietechnieken of aangepaste paradigmata te creëren* die perfect aansluiten bij een bepaald berekeningsprobleem. Hoe complexer de programmatuur wordt, hoe meer waardevol deze werkwijze is. Een programma dat geschreven is van beneden naar boven, kan men beschouwen als een opeenvolging van abstractielagen. Daarbij is de ene laag de programmeertaal van de andere. Deze manier van werken maakt Lisp een multi-paradigma programmeertaal die nog steeds actief is in de moderne programmeertechnologie. Zo werd bijvoorbeeld in 1988

CLOS toegevoegd aan Common Lisp. Dit is een uitbreiding van Lisp met het object-georiënteerde programmeerparadigma.

Het schrijven van beneden naar boven leidt natuurlijkerwijze tot *uitbreidbare software*. In de opeenvolging van abstractielagen, zal de bovenste laag fungeren als programmeertaal voor de gebruiker. Doordat de idee van uitbreidbaarheid zo diep geworteld is in Lisp, is het niet verwonderlijk dat er al diverse software gebaseerd is op Lisp zoals GNU EMACS, AUTOCAD and INTERLEAF.

Een tweede belangrijk voordeel dat schrijven van beneden naar boven met zich meebrengt, is *herbruikbare software*. De essentie van herbruikbare software is om algemene zaken te onderscheiden van specifieke. Deze scheiding is inherent aan het schrijven van beneden naar boven. De meest specifieke zaken zullen zich in de bovenste abstractielaag of programmeertaal bevinden. De onderliggende abstractielagen bieden de opportuniteit om nieuwe en andere specifieke lagen te ontwikkelen.

De meest specifieke abstractielaag uit de reeks zal over grote abstractieconcepten beschikken. Deze laag zal bijgevolg nauw aansluiten bij het uit te drukken berekeningsprobleem. Op die manier kan de *programmacode korter* zijn en wordt de *leesbaarheid bevorderd* doordat deze opbouwd is uit *concepten met een hogere uitdrukkingskracht*. Dit is precies wat Abselson & Sussman onder goede programmacode begrijpen in hun boek "Structure and Interpretation of Computer Programs" [AwJS96]. Hoog-niveau programmacode wordt immers in de eerste plaats gecreëerd door mensen. De omzetting van deze hoog-niveau programmacode naar machine-instructies is een andere kwestie. Dit idee staat weergegeven in volgend citaat.

> Programs must be written for people to read, and only incidentally for machines to execute.
>
> Abelson & Sussman, SICP, voorwoord eerste editie

Onderstaande opsomming is een overzicht van enkele belangrijke taaleigenschappen en abstractietechnieken van Lisp. Deze maken taaluitbreidingen en het 'van beneden naar boven' programmeren mogelijk.

- De taal Lisp is *dynamisch getypeerde en statisch gescoped*. Alle objecten in Lisp worden geassocieerd met *symbolen*, deze laatste kunnen deel uitmaken van *verschillende namespaces*. Concreet betekent dit dat een symboolnaam zowel een variabele als een functie kan zijn. Er zijn standaard zeven verschillende namenspaces in ANSI Common Lisp.

- Zowel de data als de operatoren beschikken over dezelfde *uniforme syntax*.
- Lisp ondersteunt onder andere het *functionele programmeerparadigma* en heeft daarmee een solide wiskundige grondslag.
- Het meest populaire concept, dat bijvoorbeeld niet beschikbaar is in de programmeertaal C, zijn *lexical closures*. Dit krachtig programmeerconcept is mogelijk omdat Lisp beschikt over hogere-orde functies.
- Een ander uniek kenmerk van Lisp is dat Lisp-programma's worden beschreven door middel van Lisp-data. Dit betekent dat je Lisp-programma's kan schrijven die Lisp-programma's genereren. Deze programma's heten *macro's* en maken fundamentele taaluitbreidingen mogelijk.
- Lisp is een *interactieve programmeeromgeving* daar we beschikken over een edit-compile-test cyclus. Op die manier wordt er geprogrammeerd in real-time.

Lisp wordt soms nog onterecht beschuldigd als een "trage" taal. Onder traag wordt begrepen dat de gecompileerde code minder efficiënt zou zijn dan bijvoorbeeld het geval is bij C of C++. We merken op dat, wanneer Lisp-programma's geschreven zijn met het oog op snelheid en gecompileerd worden met een moderne compiler, de snelheid gelijkaardig is als gelijk welke hoog-niveau programmeertaal.

4.4.2 Kwantumsimulator als taaluitbreiding in Lisp

Het is verbazingwekkend dat er tot op heden geen kwantumsimulatoren zijn ontwikkeld die voordeel halen uit de kenmerken van Lisp. Dit gaf de aanzet om als eerste *Lisp uit te breiden met de nodige abstracties om willekeurige kwantumcircuits in uit te drukken en te berekenen.* We hernemen alle typische kenmerken van Lisp uit de vorige sectie en tonen aan hoe deze bijdragen tot een krachtige programmeeromgeving voor de kwantumsimulator.

- **Programmeren in Lisp betekent de taal uitbreiden.** Dit stemt overeen met het vierde objectief, "een bestaande klassieke programmeertaal uitbreiden [...]".
- **Lisp is dynamisch getypeerd, statisch gescoped, werkt met eerste-klasse objecten en is voorzien van een garbage-collector.** De eigenschappen dragen ertoe bij dat we over een zeer rijke en krachtige programmeeromgeving beschikken om kwantumconcepten in uit te drukken.

- **We programmeren van beneden naar boven door verschillende abstractielagen te introduceren.** De belangrijkste abstractielagen zijn de lineaire algebra en de kwantumberekeningen zoals beschreven in het zevende objectief. De abstractielagen daarboven verbeteren het programmeercomfort om kwantumcircuits en berekeningen te modelleren. Deze kwestie wordt in detail behandeld in de hoofdstukken 5 en 6.

- **Lisp is een multi-paradigma programmeertaal.** We geloven niet dat er een ideaal programmeerparadigma bestaat om alle concepten van de kwantumsimulator in te beschrijven. We illustreren dit idee met een concreet voorbeeld. Het functionele programmeerparadigma leent zich uitstekend om wiskundige bewerkingen uit de lineaire algebra in uit te drukken. De implementatie van de matrixvoorstelling gebeurt met het object-georiënteerde paradigma. Dit is precies hoe Lisp de dingen ziet: kies het paradigma dat het beste aansluit bij het probleem in kwestie. Lisp buigt zich naar het probleem en niet omgekeerd.

- **Het schrijven van beneden naar boven leidt tot uitbreidbare en herbruikbare software.** Op die manier wensen we deel te nemen aan de wetenschappelijke dialoog om steeds andere en betere programmeertalen voor kwantumcircuits te ontwikkelen.

- **Lisp-programma's kunnen compacter, expressiever en bijgevolg leesbaarder zijn.** De programmacode van de eigen simulator is daar een voorbeeld van. In hoofdstukken 5 en 6 wordt de implementatie van de kwantumsimulator behandeld. Een cruciaal punt van deze hoofdstukken is onder andere deze kwestie. Verderop in dit werk spreken we over het zesde objectief.

- **Lisp is een interactieve programmeertaal.** Het is mogelijk een berekening in uitvoering stop te zetten, bepaalde functies aan te passen en te hercompileren om daarna de berekeningen verder te zetten. Dit is handig omdat kwantumberekeningen bijzonder rekenintensief kunnen zijn. Het herstarten van berekeningen dient geminimaliseerd te worden.

4.5 Bestaande kwantumsimulatoren

Er bestaat momenteel een grote variëteit aan voortreffelijke kwantumsimulatoren. We verwijzen naar het verslag van Wallace [Wal99] en Vanden Berghe [Ber04] voor een gedetailleerd overzicht. Er werd een keuze gemaakt tussen

drie uiteenlopende en toonaangevende werken, deze worden in de volgende secties behandeld.

4.5.1 Quantum Computation Language

QCL [Ö00] is een imperatieve hoog-niveau kwantumprogrammeertaal die werd ontwikkeld door de natuurkundige Bernhard Ömer in 2000 aan de technische universiteit van Wenen (Oostenrijk), twee jaar na de publicatie van zijn procedureel formalisme [Ö98]. De bedoeling was om een taal te ontwikkelen die effectief op een kwantumcomputer zou kunnen gebruikt worden. Daarom zijn er intern twee evaluatoren en geheugengebieden (telkens voor klassiek en kwantum) te onderscheiden. De experimenteermogelijkheden beperken zich tot het kunnen raadplegen van de kwantumtoestand.

Het resultaat is een statisch getypeerde, geïnterpreteerde klassieke programmeertaal voorzien van een interface om een fictieve kwantumcomputer aan te spreken. Wat betreft de efficiëntie van de simulatie scoort QCL bijzonder goed en fungeerde daarom als een bron van inspiratie voor de ontwikkeling van een eigen simulator. Bettelli [Bet03] bekritiseerde QCL daar het geen hogere-orde functies ondersteunt, wat inherente limitaties in de uitdrukkingskracht met zich meebrengt. Een kwantumoperator wordt gedefineerd door een tweede-klasse functie. Dit betekent onder andere dat een functie niet kan geassocieerd worden met een variabele zoals dat het geval is bij datatypes. Bijgevolg is het niet mogelijk om een nieuwe functie te ontwikkelen die bijvoorbeeld conditionele eigenschappen toevoegd aan een bestaande kwantumoperator.

QCL was een trendsetter in de implementatie van het QRAM-model "Quantum Random Access Memory" [Kni96]. We verklaren kort de idee achter dit model. Beschouwen we een enkel kwantumgeheugen $\mathcal{Q}$ dat bestaat uit n qubits (beschreven door 2^n amplitudes). Verschillende instanties kunnen een deelverzameling qubits $\mathcal{S} \subseteq \mathcal{Q}$ van dit kwantumgeheugen reserveren. Het toepassen van een willekeurige kwantumoperator U op de deelverzameling qubits $\mathcal{S}$ kan een invloed hebben op alle 2^n amplitudes van het kwantumgeheugen $\mathcal{Q}$, en dus ook op eventuele andere deelverzamelingen van $\mathcal{Q}$. Het QRAM-model leidt deze kwestie in goede banen. Via dergelijke modellen speculeert men eigenlijk over concepten die wel eens realiteit zouden kunnen worden.

4.5.2 Q Language

De ideeën van QCL fungeerden drie jaar later als inspiratie voor het werk van de natuurkundige Stefano Bettelli, "Toward an architecture for quan-

tum programming" [Bet03], waarmee hij de Q LANGUAGE programmeertaal introduceerde aan dezelfde universiteit als Ömer. De bedoeling was om een object-georiënteerde kwantum programmeertaal te ontwikkelen als uitbreiding van C++ met een hoog realismegehalte. Er wordt dus totaal geen rekening meer gehouden met experimenteermogelijkheden. Om dit alles mogelijk te maken, was enige speculatie over kwantum hardware noodzakelijk.

Het innovatieve aan zijn werk is de automatische omzetting van hoogniveau naar laag-niveau, universele, kwantumoperatoren (zie sectie 3.3.2) die naar een denkbeeldige kwantumprocessor kunnen gestuurd worden. Deze omzetting werd voorzien van een optimalisatie-mechanisme om te kunnen snoeien in het aantal laag-niveau kwantumoperatoren. Voor het kwantum geheugenmanagement wordt, net als bij QCL, een beroep gedaan op het QRAM-model. We merken op dat het realismegehalte wordt beperkt doordat kwantum foutcorrectie op geen enkele wijze in rekenschap is gebracht.

4.5.3 QPico

QPico [Ber04] is een uitbreiding van de klassieke functionele programmeertaal Pico [DM], die werd ontwikkeld door Frederik Vanden Berghe in 2004 aan het laboratorium voor programmeerkunde aan de Vrije Universiteit Brussel (België). Deze programmeertaal kent veel eigenschappen die allemaal terug te vinden zijn in Lisp: dynamisch typering, statische scoping, het gebruik van een uniforme syntax en overal eerst-klasse objecten. De klemtoon van de uitbreiding van Vanden Berghe licht op de expressiviteit van de kwantum programmeertaal en niet op het simulatiegehalte. De experimenteermogelijkheden zijn beperkt tot het kunnen observeren van kwantumtoestanden. De volgende opsomming schetst kort het wel en wee van QPico.

- De qubits en kwantumoperatoren zijn eerste-klasse objecten en sluiten zo nauw aan bij de klassieke Pico programmeeromgeving.
- Zeer compacte en expressieve syntax om kwantumoperatoren te definiëren, samen te stellen en toe te passen. Dit is het resultaat van de "van beneden naar boven" aanpak van programmeren.
- Kwantumregisters kunnen niet in een (willekeurige) superpositie geconstrueerd worden.
- Het is enkel mogelijk om globale metingen uit te voeren.
- Het is niet mogelijk om rechtstreeks een kwantumoperator toe te passen op een enkelevoudige qubit die deel uitmaakt van een n-qubit.

- De uitbreiding van Pico tot QPico houdt concreet in dat nieuwe native functies en syntactische suiker aan de interpreter werden toegevoegd. Bijgevolg is er geen interne klassiek-kwantum scheiding in de interpreter of de geheugenmanager.
- De simulator kent geen optimalisatietechnieken, zowel op niveau van geheugengebruik als simulatie-algoritmes.
- Ondanks alle beperkingen, merken we op dat theoretisch gezien, een willekeurige kwantumberekening kan uitgevoerd worden.

De introductie van QPico is een van de eerste pogingen om een functionele kwantum programmeertaal te ontwikkelen, na André van Tonder die in 2003 een lambda-calculus [Ton03a] en denotationele sematiek [Ton03b] heeft ontwikkeld voor een functionele kwantum programmeertaal. Hij illustreerde zijn ideeën met een eenvoudige simulatie in SCHEME.

4.5.4 Deelname aan de wetenschappelijke dialoog

Het is belangrijk om in het wetenschappelijk debat rond kwantumsimulatoren een onderscheid te maken tussen *simulatoren van de werkelijkheid* en *simulatoren als model*. Het werk van Ömer en Bettelli zijn duidelijke voorbeelden van simulatoren van de werkelijkheid. Ze willen immers een kwantum programmeertaal ontwikkelen die eventueel op een echte kwantumcomputer zou kunnen gebruikt worden. Daartoe wordt er een fictief kwantumgeheugen en processor in beschouwing genomen. Er is dus weinig tot geen experimenteerruimte voorzien.

Simulatie als model

In tegenstelling tot QCL en Q Language, proberen we in dit werk een simulator als model te ontwikkelen. De bedoeling is immers om de kwantumberekeningen te koppelen aan de lineaire algebra (objectief 7) aan de hand van leesbare, expressieve en efficiënte programmacode (objectief 3 en 6) en een flexibele experimenteeromgeving (objectief 1) te ontwikkelen. Op die manier stimuleren we het onderzoek en de ontwikkeling van nieuwe kwantumalgoritmes.

Open experimenteerruimte

Het QRAM-model is niet opgenomen in de eigen simulator omdat het niet geschikt is voor experimenteerdoeleinden. Er wordt immers voortdurend

met een globaal kwantumgeheugen $\mathcal{Q}$ gewerkt. Als gevolg van de niet-scheidbaarheid van bepaalde kwantumtoestanden (zie sectie 2.2.4), is het niet altijd mogelijk om de particuliere toestand van een deelverzameling $\mathcal{S}$ te berekenen. Dit is niet wenselijk als we bijvoorbeeld het verloop van een bepaald algoritme of foutcorrectie willen bestuderen, zoals beschreven door het eerste objectief.

De eigen simulator is voorzien van kwantumregisters, dit zijn onafhankelijke entiteiten. Verschillende kwantumregisters kunnen enkel met elkaar interfereren als ze expliciet samengesteld worden door het tensorproduct te nemen. Als gevolg van de niet-scheidbaarheid van verstrengelde qubits (zie sectie 2.2.4), is er geen algemene functie die kwantumregisters terug kan splitsen in onafhankelijke entiteiten.

Verdere exploratie van kwantum programmeertalen

Net zoals QCL, Q Language en QPico wensen we deel te nemen aan een wetenschappelijk debat rond de ontwikkeling van innovatieve hoog-niveau kwantum programmeertalen. Onze eigen simulator is een nieuwe poging om vanuit een functionele programmeertaal als Lisp, een volwaardige multi-paradigma kwantum programmeertaal te ontwikkelen. Op die manier werken we verder in dezelfde programmeerkundige geest als QPico (zie sectie 4.4.2), maar proberen we een antwoord te bieden op diverse concrete beperkingen.

- Kwantumregisters kunnen in een willekeurige superpositie geconstrueerd worden. Zie sectie 5.3.
- Kwantumoperatoren kunnen op willekeurige gebieden van qubits toegepast worden. Zie sectie 6.1.4.
- De simulator is voorzien van tal van particuliere en algemene optimalisatietechnieken. Zie secties 5.3, 6.1.3, 6.1.4 en 6.2.4.
- De taaluitbreidingen zijn geschreven met het oog op compacte en expressieve concepten om willekeurige kwantumcircuits in uit te drukken. Zie hoofdstuk 7.

We leggen de nadruk op het feit dat we ons niet willen verstikken in een vooropgelegd programmeerparadigma van een bepaalde programmeertaal. We maken daarbij komaf met de “van boven naar beneden” programmeerstijl die werd gevolgd door Ömer en Bettelli. De idee om als eerste een kwantumsimulator als taaluitbreiding in Lisp te ontwikkelen, nodigt uit tot verdere boeiende debatten rond de ontwikkeling van hoog-niveau kwantum programmeertalen.

Hoofdstuk 5

Klassieke datastructuren voor kwantuminformatie

In het vorige hoofdstuk zijn zeven objectieven beschreven waarbinnen de simulator is ontwikkeld. Binnen deze filosofie zetten we nu de bouw van kwantumregisters en operatoren uiteen. De voorstelling van kwantuminformatie op een klassieke computer kent door de kwantummechanische natuur een exponentiële groei qua geheugencomplexiteit. We gaan na welke optimalisatietechnieken gebruikt kunnen worden.

5.1 Introductie

De kwantumsimulator bestaat uit twee grote abstractielagen: klassieke berekeningen en kwantum berekeningen. De hoofdtaak van de simulator is om de kwantum berekeningen om te zetten in klassieke berekeningen, dit is de lineaire algebra. Om dit te kunnen realiseren, hernemen we eerst welke kwesties en problemen er ontstaan bij de voorstelling van qubits door middel van klassieke informatie. Het zal blijken dat het meest cruciale probleem de exponentiële groei van de ruimtecomplexiteit is. Daarom dat we in sectie 5.2.2 een datastructuur voor ijle matrices ontwikkelen. Deze geheugenefficiënte datastructuur zal later gebruikt worden om zowel qubits als kwantumoperatoren voor te stellen.

De volgende stap is om de verschillende lineaire operatoren te gaan implementeren, dit impliceert dat er vaak over (ijle) matrices zal geïtereerd moeten worden. Met het oog op het zesde objectief, het ontwikkelen van abstractiemechanismes met een hoge uitdrukkingskracht, introduceren we de `matrix-do` macro om over matrices te itereren. De kracht van deze ma-

cro wordt geïllustreerd met de implementatie van enkele vaak voorkomende lineaire operatoren.

Bovenop de lineaire algebra abstractielaag, creëren we in sectie 5.3 de abstractielaag voor kwantumberekeningen. Dit hoofdstuk is beperkt tot het voorstellen van kwantuminformatie, dit zijn qubits en kwantumoperatoren. Het simuleren van kwantum berekeningen wordt behandeld in hoofdstuk 5.

De datastructuur die qubits voorstelt, heten we kwantumregisters '`qureg`'. Het geheugengebruik wordt geoptimaliseerd door gebruik te maken van ijle matrices. Een `qureg` is, naast enige experimenteervrijheid, onderworpen aan de postulaten van de kwantummechanica. Een beknopte samenvatting van de belangrijkste kenmerken is opgenomen in sectie 5.3.1. Terug met het oog op het zesde objectief, ontwikkelen we ander iteratiemechanisme `qureg-do` specifiek voor kwantumregisters. Deze laatste maakt het mogelijk om met behulp van filters over basistoestanden te itereren die voldoen aan een bepaald bitpatroon. Het belang van deze filters zal volledig duidelijk worden in hoofdstuk 5 als we het hebben over simulatie-algoritmes.

In de laatste sectie 5.4 introduceren we een datastructuur `qop` om kwantumoperatoren voor te stellen. Deze datastructuur maakt, net als kwantumregisters, gebruik van ijle matrices om te snoeien in de exponentiële groei van de ruimtecomplexiteit.

5.2 Lineaire algebra abstractielaag

We gaan Lisp uitbreiden met een lineaire algebra abstractielaag. De bedoeling is dat we daarboven een abstractielaag voor kwantum berekeningen ontwikkelen. Dit verklaart waarom we eerst nagaan wat de kwesties en kernproblemen zijn bij het voorstellen van qubits door middel van klassieke informatie.

5.2.1 Kwesties en problematiek

Vooraleer we overgaan tot het geheugenprobleem, verklaren we eerst waarom we qubits voorstellen als toestandsvectoren en niet als densiteitsoperatoren. De laatste kwestie verklaart waarom we genoodzaakt zijn om de amplitudes numerisch voor te stellen met inbegrip van eventuele afrondingsfouten.

Toestandsvectoren versus densiteitsoperatoren

In secties 2.2 en 2.3 zijn twee verschillende formalismes voor het wiskundige raamwerk van kwantumberekeningen geïntroduceerd. Er werd een onderscheid gemaakt tussen het ideale model met toestandsvectoren en de taal

van densiteitsoperatoren. De idee bij deze laatste bestaat erin om een probabilistische component toe te voegen aan de toestandsvectoren. Op die manier is het mogelijk om 'realiseerbare' kwantumcomputers te simuleren. Deze vorm van simuleren stemt echter niet overeen met het tweede objectief (zie pagina 40). Immers, de bedoeling is om kwantumalgoritmes te simuleren en geen realiseerbare kwantumcomputers. Vandaar dat we verder gaan met het formalisme rond toestandsvectoren.

Als kantlijn merken we op dat het gebruik van densiteitsoperatoren, de geheugencomplexiteit met een factor 2^n zou vergroten daar $\rho \in \mathbb{C}^{2^n \times 2^n}$. In de praktijk zou de simulatie van realiseerbare kwantumcomputers sterk gelimiteerd zijn (in het aantal qubits) door de geheugencapaciteit van de onderliggende architectuur.

Nood aan geheugenoptimalisatie

Beschouwen we de vectorvoorstelling van een willekeurige n-qubit $|\psi\rangle$. We herhalen dat deze qubit kan beschreven worden door middel van een linaire combinatie. Wanneer we deze lineaire combinatie uitrekenen, bekomen we een kolommatrix die complexe getallen $a_i \in \mathbb{C}$ bevat met $0 \leq i < 2^n$. Een complex getal z bestaat uit twee gedeeltes $z = a + bi$ met $a, b \in \mathbb{R}$. Dit impliceert dat een n-qubit beschreven wordt door $2 \cdot 2^n$ reële getallen.

$$|\psi\rangle = \sum_{i=0}^{2^n-1} a_i |v_i\rangle = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{2^n-1} \end{pmatrix} \tag{5.1}$$

$$|v_0\rangle = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, |v_1\rangle = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \ldots, |v_{2^n-1}\rangle = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \tag{5.2}$$

Een rechtstreekse implementatie van formule (5.1) suggereert het gebruik van een array. Het grote voordeel van dergelijke datastructuur is de directe geheugentoegang in $O(1)$. Er wordt echter op voorhand een geheugengebied gereserveerd, bepaald door het aantal elementen die de array maximaal kan bevatten. Als we aannemen dat een reël getal wordt voorgesteld door bijvoorbeeld het C++ datatype `double` dat 8 bytes telt, zal een n-qubit in dat geval $\frac{2 \cdot 8 \cdot 2^n}{2^{10}}$ MB in beslag nemen. Vanaf $n > 20$ is het geheugengebruik exhuberant groot in termen van de geheugencapaciteit van een gemiddelde moderne workstation anno 2005. De simulator is voorzien van een optimalisatie die dit probleem aanpakt. Dit staat beschreven in sectie 5.2.2.

Discretisering en afrondingsfouten

Aanvankelijk werd er geprobeerd om de lineaire algebra abstractielaag uit te rusten met een Computer Algebra System (CAS). De idee is om wiskundige bewerkingen niet numerisch uit te rekenen, maar symbolisch te verwerken. Voor dergelijke symbolische verwerking kan een beroep gedaan worden op bestaande Computer Algebra Systemen (CAS). De strategie bestaat erin om de tijdscomplexiteit aanvaardbaar te houden ten koste van de ruimtecomplexiteit. De resultaten waren echter teleurstellend.

Het belangrijkste voordeel bij symbolische verwerking, is dat het toepassen van rekenregels in veel gevallen efficiënter is dan het numerisch berekenen van een wiskundige bewerking. Bij de symbolische verwerking van bijvoorbeeld $\sqrt{a}\cdot\sqrt{b} = \sqrt{ab}$, hoeft enkel fysiek ab berekend te worden. Het numerisch bepalen van deze wortelvermenigvuldiging is qua rekentijd minder efficiënt.

Het tweede voordeel is dat we niet werken met floating points. Het geheugen van een computer is eindig en irrationale getallen worden dus gediscretiseerd. Afhankelijk van de nauwkeurigheid, kan het gebeuren dat $\sqrt{2}\cdot\sqrt{2} \approx 1,41\cdot 1,41 \approx 1,98$. De symbolische verwerking heeft in dergelijke omstandigheden deze afrondingsfout niet $\sqrt{2}\cdot\sqrt{2} = \sqrt{2\cdot 2} = \sqrt{4} = 2$.

Het grootste nadeel is dat sommige wiskundige expressies niet kunnen vereenvoudigd worden. Vanaf dat ogenblik kan de ruimtecomplexiteit een explosieve groei kennen.

Hergebruik van een bestaande CAS is maar gedeeltelijk mogelijk. Dit komt omdat deze bepaalde bewerkingen uitvoeren die niet wenselijk zijn. Bijvoorbeeld, bij evaluatie van $\sqrt{24} = 2\sqrt{6}$, wordt (voor de leesbaarheid) gepoogd de expressie te verveenvoudigen. Daartoe gaat een CAS stilzwijgend een zeer onefficiënte priemfactorisatie berekenen. Dit is in context van een kwantumsimulator totaal ongewenst.

De absolute uitvoeringstijd van symbolische verwerking hangt sterk af van de technische componenten. Op een doorsnee workstation is een geheugentoegang trager dan de verwerkingskracht van de processor. M.a.w. de ruimtecomplexiteit laten toenemen ten voordele van de tijdscomplexiteit schijnt theoretisch interessant te zijn, maar gaat in de praktijk niet op. Bijgevolg werd deze poging tot optimalisatie uiteindelijk geschrapt uit de simulator. We behelpen ons met een numerische benadering zoals aangegeven door het tweede objectief op pagina 38.

5.2.2 Efficiënte matriximplementatie

Als antwoord op bovenstaand probleem ‘nood aan geheugenoptimalisatie’ werd een efficiënte matriximplementatie ontwikkeld. De idee is geïnspireerd

op de techniek van ijle matrices, die geheugenverspilling probeert te reduceren. De bedoeling is om enkel elementen van een matrix bij te houden die verschillend zijn van 0. Op die manier is het mogelijk om virtuele matrices te definiëren met zeer grote dimensies.

Deze matrices worden intern voorgesteld door middel van een hashtabel. Er werd een eenvoudige hashfunctie ontwikkeld die een matrixpositie (i, j) afbeeldt op een unieke sleutel. Alle elementen die 0 zijn worden niet opgenomen in de hashtabel. Op die manier kunnen zeer grote matrices gedefinieerd worden, en wordt de benodigde geheugenruimte geminimaliseerd. Daarnaast werd een speciale macro ontwikkeld die enkel over elementen itereert die verschillend zijn van 0, op die manier wordt er geen rekentijd verspild.

De efficiëntie is sterk beïnvloed door het aantal keer dat de hashtabel herschaald moet worden tijdens zijn levensduur. Wanneer een hashtabel volzet raakt en moet groeien, wordt er kostbare rekentijd verspild. De bedoeling is om onderstaande parameters zodanig in te schatten zodat herschalingen geminimaliseerd worden.

- initiële grootte van de hashtabel;
- de schalingsfactor van de hashtabel die toegepast wordt bij overbezetting;
- de drempelwaarde die aangeeft of een hashtabel overbezet is of niet.

Het is niet mogelijk om parameters te specificeren die zich ideaal gedragen bij een willekeurige kwantumberekening. Elk kwantumcircuit kent impliciet zijn eigen ideale parameterwaardes voor het herschalen van de hashtabel. Het vergt dieptezicht in een kwantumcircuit om deze parameters nauwkeurig te kunnen inschatten.

De ijle matrixvoorstelling is geïmplementeerd als een Lisp-structure. De definitie wordt geïllustreerd door onderstaand vereenvoudigd code-extract.

```
(defstruct (matrix
    (:constructor make-matrix (cols rows)))
  "create a cols x rows matrix"
  cols rows (contents (make-hash-table) :read-only t))
```

De functie `(get-element matrix entry)` geeft een element uit `matrix` terug gespecificeerd door `entry`. Matrixelementen kunnen gewijzigd worden door `get-element` te combineren met de `setf` functie. Een `entry` is een datastructuur die een positie (i, j) binnen een matrix aanwijst. De Lisp-structure van deze datastructuur ziet er als volgt uit. Merk op dat indices steeds starten vanaf 0.

```
(defstruct (entry (:constructor make-entry (col row)))
  "indicates position: col [0...cols] and row [0...rows]"
  col row)
```

Het manueel definiëren van matrices kan gerealiseerd worden met de `init-matrix` macro. Onderstaand code-extract is de matrixdefinitie van de Hadamard kwantumoperator die inwerkt op een 2-qubit.

```
(let ((a (/ 1 2)))
  (init-matrix
    ((a  a  a  a)
     (a -a  a -a)
     (a  a -a -a)
     (a -a -a  a))))
```

5.2.3 Itereren over matrices

Tijdens het uitvoeren van kwantumberekeningen, zal er regelmatig over matrices geïtereerd moeten worden. Denk maar aan de vermenigvuldiging van een reël getal met een matrix, een matrixvermenigvuldiging of het tensorproduct. Daarom is het wenselijk om een abstractiemechanisme te ontwikkelen met een hoge uitdrukkingskracht om over matrices te kunnen itereren. De abstractie heeft als doelstelling om codeduplicatie van vaak voorkomende patronen te reduceren, de leesbaarheid te bevorderen en bijgevolg de gebruiksvriendelijkheid te verhogen.

Om dit te kunnen concretiseren, gaan we na waar de problemen zich precies situeren. We nemen eerst een prototypisch code-extract onder de loep. Er wordt nagegaan waar de uitdrukkingskracht tekort schiet. Op basis van de vastgestelde kritieken worden criteria gespecificeerd waaraan de eigen iteratiemechanismes moeten voldoen.

Basiscriteria voor eigen iteratiemechanisme

We analyseren een code-extract van de Java bibliotheek JAMA [MN] die een matrixvermenigvuldiging implementeert. De kritiek op deze programmacode is dat het veelvuldig gebruik van indices en geneste lussen, de hoeveelheid programmacode sterk doet stijgen en het geheel onleesbaar maakt. We beweren dat de uitdrukkingskracht hier laag ligt. Dit is in strijd met het zesde objectief beschreven op pagina 45.

```
public Matrix times (Matrix B) {
   Matrix X = new Matrix(m,B.n);
```

```
   double[][] C = X.getArray();
   double[] Bcolj = new double[n];
   for (int j = 0; j < B.n; j++) {
      for (int k = 0; k < n; k++) {
         Bcolj[k] = B.A[k][j];
      }
      for (int i = 0; i < m; i++) {
         double[] Arowi = A[i];
         double s = 0;
         for (int k = 0; k < n; k++) {
            s += Arowi[k]*Bcolj[k];
         }
         C[i][j] = s;
      }
   }
   return X;
}
```

De matrixvermenigvuldiging is in dit code-extract gerealiseerd door geneste `for` lussen. Dit impliceert dat er verschillende hulpvariabelen gedefinieerd moeten worden zoals `Bcolj`, `n`, `m`, `i`, `j` en `k`. We leiden hieruit onderstaande eerste criteria af om een eigen iteratiemechanisme te ontwikkelen.

Criterium 1 *Het eigen onderhoud (initialisatie en updaten) van hulpvariabelen moet geminimaliseerd worden.*

Het gebruik van het iterator design pattern zou een oplossing kunnen zijn om `for`-lussen te abstraheren. We beweren dat dit in Java automatisch zou leiden tot langere programmatekst omdat we 'van boven naar beneden' programmeren (zie sectie 4.4.1) binnen het object-georiënteerde programmeerparadigma. Bijvoorbeeld het gebruik van een iterator in Java betekent concreet de constructie van een iterator, in elke iteratiestap hetvolgende element ophalen en telkens verifiëren of er nog elementen volgen. We vatten onze eis samen in volgend criterium.

Criterium 2 *Het mechanisme moet kort en krachtig zijn.*

De implementatie van lineaire operatoren zal vaak te kampen hebben met steeds terugkerende iteratiepatronen. In het stukje Java-code zien we bijvoorbeeld al herhalingen van `for`-lussen die opvallend veel gelijkenissen kennen. Dit leidt tot code-duplicatie, daarom wensen we een mechanisme te ontwikkelen dat krachtig genoeg is om zoveel mogelijk herbruikbaar te zijn.

Criterium 3 *Het mechanisme moet voldoende algemeen zijn om in zoveel mogelijk contexten toepasbaar te zijn.*

Taaluitbreidingen in Lisp met macro's

De macro's voorzien in de taal Lisp (zie sectie 4.4.1) ontlenen zich uitstekend om de drie criteria te realiseren zonder noewenswaardige efficiëntie in te boeten. We concretiseren de bovengenoemde drie criteria. We willen een iteratiemechanisme ontwikkelen dat:

1. indices zoveel mogelijk afschermt;
2. steeds terugkerende patronen minimaliseert en
3. conditioneel kan itereren.

De macro `matrix-do`, waarvan de interface hieronder staat weergegeven, vervult de drie bovenstaande voorwaarden. In sectie 5.3.3 wordt nog ander iteratiemechanisme voorgesteld om intelligent over de basistoestanden van een kwantumregister te kunnen itereren.

```
(matrix-do
  ((matrix entry element &key condition skip-zero result)
  &body body)
```

De `matrix-do` itereert over een gegeven `matrix`. Daarbij worden twee symbolen `entry` en `element` gedefinieerd die bij elke iteratiestap respectievelijk de positie en het corresponderende matrixelement bevatten. Optioneel kan er een conditie gespecificeerd worden, dit is een predikaatfunctie met twee parameters `entry` en `element`. Wanneer deze `nil` retourneert, wordt de iteratie onmiddellijk stopgezet. De `skip-zero` is een optimalisatie die ingeschakeld kan worden in berekeningen waarbij enkel elementen verschillend van 0 relevant zijn. De return-waarde van deze macro is standaard `t`, tenzij de iteratie werd onderbroken doordat `condition` niet meer waar was of er een andere return-waarde werd gespecificeerd met `result`.

5.2.4 Implementatie van lineaire operatoren

We illustreren de concrete werking van de macro `matrix-do` aan de hand van een paar eenvoudige voorbeelden uit de lineaire algebra.

Adjunct van een matrix

De functie `get-adjoint` berekent de adjunct van `matrix`. De adjunct is de samenstelling van een matrixtransponatie (kolommen en rijen omwisselen) en het nemen van de complex toegevoegde waarde van elk matrixelement. De implementatie ziet er als volgt uit.

```
(defun get-adjoint (matrix)
  "get the adjoint of a matrix"
  (let ((result (make-matrix (matrix-cols matrix)
                             (matrix-rows matrix))))
    (matrix-do (matrix entry element)
      (setf (get-element result (transpose-entry entry))
        (conjugate element)))
    result))
```

De functie `transpose-entry` wisselt de rol van kolom en rij, de complexe toegevoegde waarde van een matrixelement wordt berekend met `conjugate`.

Matrixvermenigvuldiging

De implementatie van de matrixvermenigvuldiging maakt gebruik van de krachtige `loop` macro die deel uitmaakt van ANSI Common Lisp [Lim].

```
(defun multiply-matrices (matrix1 matrix2)
  "multiply two matrices straightforward"
  (let ((result (make-matrix (matrix-cols matrix2)
                             (matrix-rows matrix1))))
    (matrix-do
      (result entry element)
      (setf (get-element result entry)
        (loop for k from 0 below (matrix-cols matrix1) sum
          (* (get-element matrix1
                          (make-entry k (entry-row entry)))
             (get-element matrix2
                          (make-entry (entry-col entry) k))))))
    result))
```

Tensorproduct

De definitie van deze lineaire bewerking staat in bijlage A.1.5.

```
(defun tensor (matrix1 matrix2)
  "perform tensor product"
  (let ((result (make-matrix (* (matrix-cols matrix1)
                                (matrix-cols matrix2))
                             (* (matrix-rows matrix1)
                                (matrix-rows matrix2)))))
    (matrix-do
      (matrix1 entry1 element1)
      (matrix-do
        (matrix2 entry2 element2)
          (setf (get-element
                 result
                 (make-entry
                  (+ (* (entry-col entry1) (matrix-cols matrix2))
                     (entry-col entry2))
                  (+ (* (entry-row entry1) (matrix-rows matrix2))
                     (entry-row entry2))))
            (* element1 element2))))
    result))
```

5.3 Kwantumregisters

Bovenop de lineaire algebra abstractielaag uit de vorige sectie, bouwen we een abstractielaag om kwantum informatie voor te stellen. Een kwantumregister is een datastructuur die een qubit voorstelt. We beginnen de bespreking van deze datastructuur met een beknopte herhaling van de verschilpunten tussen klassieke en kwantum informatie. We eindigen deze sectie met de introductie van een ander iteratiemechanisme om over de basistoestanden van een kwantumregister te itereren.

5.3.1 Klassieke versus kwantum geheugenregisters

De simulator beschikt over twee denkbeeldige geheugengebieden (klassiek en kwantum) die conceptueel sterk verschillend zijn door de fysische achtergrond, respectievelijk Newtoniaanse fysica en kwantummechanica. Het *klassieke geheugengebied* omvat alle ingebouwde datatypes beschikbaar in ANSI Common Lisp [Lim] en alle gebruikers-gedefinieerde uitbreidingen. Deze laatste zijn steeds afgeleid van de ingebouwde datatypes.

Het kwantumregister is het enige primitieve datatype dat gedefinieerd is in het denkbeeldige *kwantum geheugengebied*. Bij de constructie van een

kwantumregister wordt in Lisp een symbool van het type kwantumregister gedefinieerd die een qubit voorstelt. Als gevolg van de kwantummechanische natuur heeft een kwantumregister niet dezelfde gedragskenmerken als een klassiek geheugenregister. We vatten de belangrijkste verschilpunten samen in onderstaande opsomming.

- Het gedrag van een klassiek geheugenregister wordt beschreven door de klassieke wetten van de Newtoniaanse fysica. Bij een kwantumregister gebeurt dit door de postulaten van de kwantummechanica.
- De toestand van een klassiek geheugenregister is altijd eenduidig (0 of 1). Een kwantumregister bevindt zich in een continuüm van basistoestanden, superposities genoemd.
- Klassieke geheugenregisters zijn omkaderd door een deterministisch berekeningssysteem. Dit betekent dat er geen toevalscomponent in het systeem zit. Kwantumregisters maken deel uit van een niet-deterministisch berekeningssysteem. Concreet betekent dit dat er met elke superpositie een bepaalde probabiliteit geassocieerd is.
- Als gevolg van de niet-klonen stelling (zie sectie 2.2.4) kunnen kwantumregisters niet gekopieerd worden. Klassieke geheugenregisters kunnen steeds gekopieerd worden.
- De toestand van een klassiek geheugenregister kan steeds geraadpleegd worden. Het observeren van een kwantumregister houdt een meting in. Dit betekent dat zijn toestand dan onherroepelijk vervalt in één van de mogelijke basistoestanden.
- Wanneer de amplitudes van een kwantumregister (dat een n-qubit voorstelt) verstrengeld zijn, is het niet meer mogelijk om dit kwantumregister te beschrijven als een tensorproduct van n enkelvoudige kwantumregisters (zie sectie 2.2.4).

Om het eerste objectief (zie pagina 38) te kunnen realiseren, dit is de flexibele experimenteeromgeving, is het toch toegestaan om kwantumregisters te observeren zonder dat deze gemeten worden of kwantumregisters te kopiëren.

5.3.2 Implementatie van kwantumregister-simulatie

De simulator beschrijft een kwantumregister op implementatieniveau met klassieke informatie. Hernemen we de vectorvoorstelling van een willeurige

n-qubit $|\psi\rangle$ voorgesteld in formules (5.1) en (5.2). $|\psi\rangle$ is in feite een kolommatrix met 2^n complexe getallen die voldoen aan de normalisatie conditie. We snoeien in de exponentiële groei van de ruimtecomplexiteit door gebruik te maken ijle matrices zoals beschreven in voorgaande sectie 5.2.2.

In werkelijkheid zou $|\psi\rangle$ kunnen voorbereid worden in een willekeurige begintoestand. Een voldoende flexibel systeem biedt de mogelijkheid om verschillende initialisatiemechanismes te definiëren voor een kwantumregister. Onderstaand voorbeeld geeft aan dat dit een belangrijke bijdrage kan leveren tot de efficiëntie van de simulatie.

Voorbeeld 5 *Doorgaans wordt een n-qubit $|\psi_1\rangle$ in de basistoestand $|0\rangle$ geinitialiseerd. Men spreekt in deze situatie over een afgekoelde kwantumtoestand.*

$$|\psi_1\rangle = 1|0\rangle + 0|1\rangle + 0|2\rangle + \ldots + 0|2^{n-1}\rangle$$

Wanneer we nu de Hadamard operator willen toepassen op alle qubits van $|\psi_1\rangle$, moeten we volgende rekenintensieve formule berekenen.

$$|\psi_2\rangle = H^{\otimes n}|\psi_1\rangle$$

Het is echter ook mogelijk om een qubit voorafgaandelijk in een specifieke begintoestand te prepareren. We hernemen bovenstaand voorbeeld en berekenen de begintoestand rechtstreeks.

$$|\psi_3\rangle = \sum_{i=0}^{2^{n-1}} \frac{1}{\sqrt{2^n}}|i\rangle = |\psi_2\rangle$$

Onderstaande structure is een extract van de `qureg` structure. Bij de constructie van een kwantumregister, dient de grootte `size` van de qubit gespecificeerd te worden. Deze variabele wordt gebruikt om de overeenkomstige `matrix` te construeren. Optioneel kan er `init-fn` meegegeven worden. Deze functie kan rechtstreeks de matrixelementen van het kwantumregister initialiseren. Om het onderscheid tussen kwantum en klassiek te accentueren, wordt de symboolnaam van kwantumregisters steeds voorzien van underscores zoals in bijvoorbeeld `_qureg_`.

```
(defstruct (qureg
  (:constructor make-qureg
     (size &optional (init-fn #'identity)
           &aux (matrix
                  (let ((result (make-matrix 1 (exp2 size))))
                    (funcall init-fn result) result)))))
  (size :read-only t) (matrix :read-only t))
```

We bespreken kort enkele handige hulpfuncties van het kwantumregister.

- `(qureg-size _qureg_)` retourneert de grootte van het kwantumregister, dit is het aantal qubits.
- `(amplitude-count _qureg_)` retourneert het aantal amplitudes.
- `(get-amplitude _qureg_ basis)` retourneert de amplitude geassocieerd met de basistoestand `basis`. In combinatie met `setf` kunnen amplitudes gewijzigd worden. Door deze kunstmatige ingreep, kan een eventuele normalisatie noodzakelijk zijn.
- `(transform-qureg matrix)` transformeert een kolommatrix met 2^n elementen naar een kwantumregister van grootte n.

De Hadamard initialisatie-functie zoals beschreven in voorbeeld 5 ziet er als volgt uit. Het kwantumregister wordt dan gedefinieerd met de expressie `(make-qureg n (hadamard-init))`.

```
(defun hadamard-init ()
  "return hadamard init-fn for quantum register"
  (lambda (matrix)
    (matrix-do (matrix entry element)
     (setf (get-element matrix entry)
       (/ 1 (sqrt (matrix-rows matrix)))))))
```

Daarnaast werd ook de initialisatiefunctie `standard-init` ontwikkeld om een kwantumregister te initialiseren in een specifieke basistoestand `basis`.

```
(defun standard-init (&optional (basis 0))
  "return standard init-fn for quantum register"
  (lambda (matrix)
    (setf (get-element matrix (make-entry 0 basis)) 1)))
```

De macro `init-qureg` laat toe manueel de 2^n amplitudes a_i uit formule (5.1) te specificeren in oplopende volgorde $i = 0, i = 1, \ldots, i = 2^n - 1$. Wanneer de normalisatie conditie niet voldaan is, wordt deze automatisch toegepast. Onderstaand voorbeeld is de initialisatie van $1/2|0\rangle + \sqrt{3}/2|1\rangle$.

```
(init-qureg ((/ 1 2) (/ (sqrt 3) 2)))
```

Kwantumregisters zijn onafhankelijke kwantummechanische systemen. Dit betekent dat ze niet interfereren met andere kwantumregisters. Wanneer een kwantumoperator inwerkt op verschillende kwantumregisters, kunnen kwantumregisters samengesteld worden met behulp van het tensorproduct. Dit wordt geïllustreerd in onderstaand voorbeeld.

```
(let ((_a_  (init-qureg ((/ 1 2) (/ (sqrt 3) 2))))
      (_b_  (make-qureg 2 (hadamard-init)))
      (_c_  (make-qureg 3 (standard-init))))
  (tensor-items _a_ _b_ _c_))
```

5.3.3 Handig itereren over kwantumregisters

We breiden Lisp uit met een ander iteratiemechanisme `qureg-do` om over de basistoestanden van een kwantumregister te itereren. Dit mechanisme voldoet aan dezelfde criteria, zoals beschreven in sectie 5.2.3, als de `matrix-do`. De kernidee van `qureg-do` is dat we in bepaalde omstandigheden slechts over een deelverzameling basistoestanden willen itereren die voldoen aan een bepaald bitpatroon. De gebruiksvriendelijkheid en efficiëntie waarmee de `qureg-do` macro dit mogelijk maakt, biedt ons een uitzonderlijk hoge uitdrukkingskracht.

In het volgende hoofdstuk over simulatie-algoritmes zal de praktische relevantie van dit iteratiemechanisme in detail verklaard worden. Om toch al een tipje van de sluier te lichten, geven we in voorbeeld 6 aan hoe de conditionele Pauli-Z kwantumoperator efficiënt kan berekend worden met behulp van dit iteratiemechanisme.

Voorbeeld 6 *Veronderstel dat we een conditionele Pauli-Z operator toepassen op een 3-qubit met de eerste qubit als conditie en de derde qubit als doel. Dit kan eenvoudig berekend worden door alle amplitudes geassocieerd met de basistoestand* $|1?1\rangle$ *te vermenigvuldigen met* -1*. De wiskundige achtergrond van deze optimalisatie staat in sectie 6.1.3.*

De uiteenzetting van de `qureg-do` macro verloopt als volgt. Eerst verklaren we het verband tussen de decimale en binaire voorstellingswijze van toestandsvectoren. Vooraleer we daarna overgaan tot de bespreking van het algoritme dat `qureg-do` mogelijk maakt, gaan we na hoe we bitpatronen kunnen voorstellen.

Decimale en binaire notatiewijze voor toestandsvectoren

Er zit een kleine subtiliteit verborgen in de notatie van qubits als toestandsvectoren. We hernemen de algemene formule van een n-qubit $|\psi\rangle$ en laten

de indices steeds starten vanaf 0. Formule (5.3) gebruikt de decimale notatie $|i\rangle$ voor de superposities.

$$|\psi\rangle = \sum_{i=0}^{2^{n-1}} a_i|i\rangle = a_0|0\rangle + a_1|1\rangle + a_2|2\rangle + \ldots + a_{2^{n-1}}|2^{n-1}\rangle \qquad (5.3)$$

Deze notatiewijze is wiskundig volledig correct, maar is *weinig betekenisgevend*. Daarom wordt vaak de binaire voorstelling $x = x_0x_1 \ldots x_{n-1}$ met $x_0, x_1, \ldots, x_{n-1} \in \{0, 1\}$ gebruikt die overeenstemt met een decimale waarde i. Dit idee staat weergegeven in onderstaande formule.

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} a_x|x\rangle = a_{0\ldots0}|0\ldots0\rangle + a_{0\ldots1}|0\ldots1\rangle + \ldots + a_{1\ldots1}|1\ldots1\rangle \quad (5.4)$$

De termen van bovenstaande sommatie omvatten alle mogelijke bitwoorden x van lengte n.

Itereren over bitpatronen

Beschouwen we een n-qubit $|\psi\rangle$ met basistoestanden die de binaire vorm $|v_{p_0}v_{p_1}\ldots v_{p_{n-1}}\rangle$ kennen waarbij $v_{p_i} \in \{0, 1\}$. Merk op dat we aannemen dat indices steeds starten vanaf 0. Een bitpatroon is een datastructuur die aangeeft over welke basistoestanden geïtereerd mag worden op basis van een verzameling condities C. Een enkele conditie $c_i = (p_i, v_i)$ is een paar dat een bitpositie p_i en een bitwaarde v_i bevat. We illustreren de werking met een eenvoudig voorbeeld.

Voorbeeld 7 *Beschouwen we een 4-qubit $|\psi\rangle$ en een verzameling condities $C = \{(0, 1), (2, 1)\}$. De geassocieerde superposities van dit bitpatroon zijn $a_{1010}|1010\rangle$, $a_{1011}|1011\rangle$, $a_{1110}|1110\rangle$ en $a_{1111}|1111\rangle$.*

We gaan na hoe we dit concept kunnen implementeren. Een eerste kwestie is de symbolische voorstellingswijze van wat hierboven een bitpatroon wordt genoemd. Een goed idee zou kunnen zijn om dit te formuleren als een query $|1?1?\rangle$. Deze beschrijving van de verzameling C is enkel haalbaar wanneer de qubit voldoende klein is. Het formuleren van een query voor een 100-qubit waarbij de eerste en de laatste qubit 1 dienen te zijn, zou een verlies van rekentijd en geheugengebruik impliceren.

Deze laatste beperking suggereert het gebruik van conditieparen $c_i = (p_i, v_i)$ zoals gebruikt in voorbeeld 7. Maar ook deze werkwijze heeft te kampen met een subtiel probleem. Veronderstellen we een 100-qubit waarbij

we over alle basistoestanden willen itereren waarbij de eerste 50 bitwaardes 1 zijn. Dit zou een expliciete generatie van 50 conditieparen vereisen met opnieuw een verlies van rekentijd en geheugengebruik als gevolg.

Bovenstaande problematiek werd als volgt opgelost. Er zijn twee verschillende voorstellingswijzes voor bitpatronen ontwikkeld. Beide mechanismes construeren conditie-paren die we `filter-item`'s noemen. Het verschil tussen de twee zit hem in hoe deze `filter-item`'s tot stand zijn gekomen.

De `normal-filter` wordt geconstrueerd op basis van een lijst met conditieparen $c_i = (p_i, v_i)$. Deze manier van werken is ondere andere interessant als er manueel weinig condities gespecificeerd moeten worden. Bijvoorbeeld het filteren van een 4-qubit op basistoestand $|1?1?\rangle$ gebeurt met `(make-normal-filter '((0 1) (2 1)))`.

De `lazy-filter` veronderstelt dat het bitpatroon niet manueel wordt gedefiniëerd, maar wordt bepaald door een ander berekeningsproces. Dergelijke filter wordt geconstrueerd op basis van een gegeven bitwoord x en een selectiemechanisme `selection` dat relevante bits uit x aanwijst.

Met een `selection` kunnen de positie-indices p_i van alle conditieparen gespecificeerd worden. Het is mogelijk om particuliere indices p_i aan te duiden of een aaneensluitende reeks van indices p_i. Aaneensluitende reeksen van indices worden niet expliciet in het geheugen gegenereerd. Door middel van *uitgestelde evaluatie* worden indices pas gegenereerd wanneer daar om gevraagd wordt (vandaar ook de naam `lazy-filter`). De selectiecriteria van een `selection` worden voorgesteld door middel van geneste lijsten. We illustreren dit met twee voorbeelden. Indices starten steeds vanaf 0.

Voorbeeld 8 *De selectie van de eerste ($p_i = 0$) en de laatste qubit ($p_i = 99$) van een 100-qubit:*

```
(make-selection '(0 99))
```

De selectie van de eerste ($p_i = 0$), derde ($p_i = 2$), 20ste ($p_i = 19$) en laatste 50 bits ($50 \leq p_i \leq 99$) van een 100-qubit:

```
(make-selection '(0 2 19 (50 99)))
```

De structures `normal-filter` en `lazy-filter` zijn afgeleid van dezelfde structure `filter`. Ze beschikken beide over een `next-filter-item` methode die hetvolgende `filter-item` genereert en een `empty-filter-p` methode die aangeeft of er nog `filter-item`'s resteren. Onderstaande vereenvoudigde code-extracten illustreren de interface en de bouw van `filter-item`, `normal-filter` en `lazy-filter`.

```
(defstruct (filter-item
            (:constructor make-filter-item (pos bit)))
  pos bit)

(defstruct (normal-filter
            (:include filter)
            (:constructor make-normal-filter (alist)))
  alist)

(defstruct (lazy-filter
            (:include filter)
            (:constructor make-lazy-filter
                          (selection bit-vector)))
  selection bit-vector)
```

De tweede kwestie is om, gegeven een symbolische beschrijving van de filter, de relevante basistoestanden te bepalen. De naïeve oplossingsmethode is om de condities toe te passen op alle basistoestanden. Er bestaat echter een alternatieve algoritmische oplossingsmethode voor dit probleem gebaseerd op rekenkundige kenmerken van binaire getallen.

Onderstaande pseudocode toont aan hoe de `filter-item`'s niet als conditie worden gebruikt, maar als parameters van een *staartrecursieve rekenkundige bewerking*. De functie `filter-loop` retourneert de gefilterde decimale indices $|i\rangle$ van een willekeurige n-qubit. Dit algoritme werkt enkel als de `filter-item`'s gesorteerd zijn volgens oplopende positie-index. De betekenis van de verschillende variabelen staat verklaard in onderstaande opsomming. Als ondersteuning van de tekst zijn de bitwoorden van een 4-qubit opgenomen in tabel 5.1.

- `group-length` - Voor een positie-index p_i is de waarde van deze variabele 2^{n-i-1}. De informele betekenis van deze variabele is de groepslengte waarmee de 0-bit en 1-bit zich opvolgen met betrekking tot een bepaalde kolom p_i. Bijvoorbeeld, in kolom p_1 is de groepslengte 4 omdat er achtereenvolgens $4 \times$ 0-bit is gevolgd door $4 \times$ 1-bit.

- `group-max` - Dit is de eindindex van een groep met lengte `group-length`. We nemen aan dat alle indices starten vanaf 0. De waarde van deze variabele is bijgevolg `group-length`-1.

- `start` - Dit is de beginindex van de iteratie. Wanneer het eerste `filter-item` filtert op de 0-bit, is de beginindex 0. In het andere geval is de beginindex gelijk aan `group-length`.

- `jump` - Deze variabele geeft aan hoeveel indices overgeslagen moeten worden om over een groep van 0-bits en 1-bits te springen. Dit is gelijk aan 2×`group-length`.
- `offset` - Dit is de beginindex van een bepaalde recursieve oproep.
- `max` - Deze index geeft samen met `offset` het domein van indices aan waar de recursieve oproep op filtert.

```
filterloop(quantum-register filters)
  inner-filterloop(offset max)
    filter = next-filter-item(filters)
    p = filter-item-pos(filter)
    v = filter-item-bit(filter)
    group-length = 2^(n - p - 1)
    group-max = group-length - 1
    start = if (p = 1) then group-length else 0
    jump = 2 * group-length

    for i from (offset + start) to max by jump
      if (empty-filters? filters) then
        for j from 0 to group-max
          print i + j
      else
        inner-filterloop(i, (i + group-max))

  n = size(quantum-register)
  amplitudes = 2^n
  inner-filterloop(0, (amplitudes - 1))
```

De idee uit de pseudocode van de functie `filterloop` zit verwerkt in de macro `qureg-do`. De interface staat hieronder weergegeven.

```
(qureg-do
  ((qureg basis amplitude &optional filter-pattern)
  &body body)
```

`qureg-do` itereert over de amplitudes van alle basistoestanden van het kwantumregister `qureg`. Daarbij worden twee symbolen `basis` en `amplitude` gedefinieerd die bij elke iteratiestap respectievelijk de decimale basisindex en de corresponderende amplitude bevatten. De basisindex stemt overeen met de variabele i en de amplitude met de variabele a_i uit formule (5.3). Optioneel

p_0	p_1	p_2	p_3
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Tabel 5.1: bitwoorden van 16 basistoestanden (4-qubit)

kan er een `filter-pattern` gespecificeerd worden die basistoestanden filtert op basis van een bitpatroon. Deze laatste kan geconstrueerd worden met behulp van `normal-filter` of `lazy-filter`.

In sectie 5.2.3 werden drie basiscriteria geïntroduceerd voor het ontwikkelen van eigen iteratiemechanismes. We tonen aan dat de macro `qureg-do` ook voldoet aan deze criteria.

1. Alle hulpindices (`group-length`, `group-max`, `group-length`, ...) zijn volledig afgeschermd. Alleen de meest functionele indices (`basis` en `amplitude`) zitten verwerkt in de interface van de macro.

2. Met betrekking tot filters is een `selection`-mechanisme ontwikkeld om particuliere bits of aaneensluitende bits aan te wijzen. De generatie ervan gebeurt door middel van uitgestelde evaluatie.

3. Het concept van filters is een krachtig wapen om verschillende simulatie-algoritmes leesbaar en bondig te kunnen formuleren.

5.4 Kwantumoperatoren

Een kwantumoperator is in feite een unitaire matrix. De formele definitie van dit soort matrices staat in formule (2.5) op pagina 9. De structure `qop` implementeert de kwantumoperator datastructuur. Als interne voorstelling worden ijle matrices gebruikt zoals beschreven in sectie 5.2.2. Op die manier geniet de kwantumoperator datastructuur ook van de geheugenoptimalisatie. We spreken af dat symbolen die kwantumoperatoren voorstellen, omringt worden door liggende streepjes `-op-` om de leesbaarheid te bevorderen.

5.4.1 Automatische specificatie

Een `qop` kan op twee verschillende manieren geconstrueerd worden. Een kwantumoperator kan gedefinieerd worden met `transform-qop` op basis van een reeds bestaande unitaire matrix. Deze werkwijze ontleendt zich uitstekend om kwantumoperatoren automatisch te genereren. We illustreren dit met een concreet voorbeeld. Onderstaande functie `qft-qop` genereert een kwantumoperator die het gedrag van de discrete Fourier transformatie voor een kwantumregister van grootte `size` implementeert.

```
(defun qft-qop (size)
  "generate a QFT-operator of the required size"
  (let* ((n (exp2 size))
         (result (make-matrix n n))
         (w (complex (cos (/ (* 2 pi) n))
                     (sin (/ (* 2 pi) n)))))
    (matrix-do
      (result entry element)
      (setf (get-element result entry)
        (/ (expt w (mod (* (entry-col entry)
                           (entry-row entry)) n))
           (sqrt n))))
    (transform-qop result)))
```

5.4.2 Manuele specificatie

Een andere mogelijkheid is om de verschillende elementen van de kwantumoperator manueel te specificeren met `init-qop`. De syntax is in dit geval dezelfde als bij `init-matrix` uit sectie 5.2.2. Als voorbeeld staat hieronder de definitie van de Hadamard kwantumoperator.

```
(defparameter -h-
  (let ((h (/ 1 (sqrt 2))))
    (init-qop ((h h) (h (- h))))))
```

5.4.3 Samenstellen van kwantumoperatoren

Kwantumoperatoren kunnen expliciet samengesteld worden met dezelfde (generieke) functie `tensor-items` zoals bij kwantumregisters.

```
(let ((-i- (init-qop ((1 0) (0 1))))
      (-x- (init-qop ((0 1) (1 0))))
      (-z- (init-qop ((1 0) (0 -1)))))
  (tensor-items -i- -x- -z-))
```

De hulpfunctie `tensor-n` is interessant wanneer het tensorproduct een aantal keer herhaald wordt met dezelfde kwantumoperator. Onderstaand voorbeeld is een dimensievergroting van de eenheidsoperator I.

```
(let ((-i- (init-qop ((1 0) (0 1)))))
  (tensor-n -i- 5))
```

Hoofdstuk 6

Simulatie-algoritmes voor kwantumberekeningen

Een kwantumberekening wordt beschreven door middel van symbolische circuits. Er wordt nagegaan hoe willekeurige kwantumcircuits kunnen gesimuleerd worden door middel van klassieke berekeningen. Om dit te realiseren, bestaan er uiteenlopende berekeningswijzes en optimalisatietechnieken. Op basis van de zeven objectieven uit hoofdstuk 4, gaan we na welke het beste aansluiten bij de ontwikkeling van een eigen simulator.

6.1 Kwantumberekeningen

6.1.1 Concept

In onderstaande samenvatting staan de belangrijkste kenmerken van kwantumoperatoren en -berekeningen opgesomd die de voorbije hoofdstukken zijn behandeld. In de volgende sectie gaan we na hoe we deze concepten kunnen implementeren.

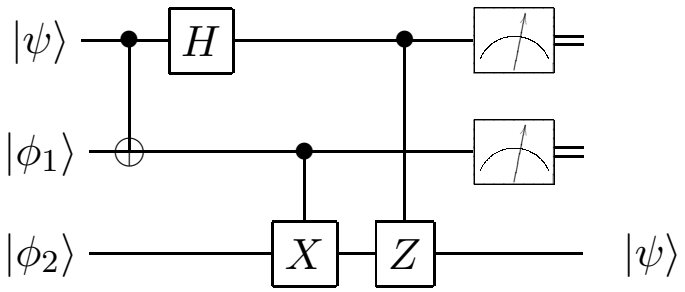
- Een kwantumberekening is een samenstelling van verschillende kwantumoperatoren (gewone en conditionele).
- De beschrijving van een kwantumberekening gebeurt voor praktische doeleinden door middel van kwantumcircuits.
- Een kwantumoperator die inwerkt op een n-qubit is een vierkante unitaire matrix met dimensie $2^n \times 2^n$.

- Elke kwantumberekening, met uitzondering van de meting, is omkeerbaar.
- Het aantal invoerqubits van een kwantumoperator is steeds gelijk aan het aantal uitvoerqubits.
- Een opeenvolging van kwantumoperatoren $O_1, O_2, \ldots, O_n$ met gelijke dimensies kan beschreven worden door een samengestelde kwantumoperator $O_{tot} = O_n \times \ldots O_2 \times O_1$ met dezelfde dimensie.

6.1.2 Rechtstreekse implementatie

Bij deze aanpak wordt voor een gegeven kwantumberekening, een samengestelde kwantumoperator berekend. Deze operator wordt dan toegepast op de invoerqubits. We illustreren deze werkwijze op basis van een concreet voorbeeld, namelijk kwantum teleportatie. We gaan na of deze werkwijze in overeenstemming is met de objectieven uit hoofdstuk 4.

Voorbeeld aanpak: kwantum teleportatie



Figuur 6.1: kwantum teleportatie circuit

We hernemen het kwantum teleportatie voorbeeld uit sectie 3.3.3. Merk op dat we hier met een ander kwantumcircuit werken. De essentie van kwantum teleportatie is hier verloren gegaan. Immers, om de conditionele operatoren op het einde te kunnen toepassen, mogen Alice en Bob niet gesplitst zijn. In dit voorbeeld is deze kwestie niet relevant. We gebruiken het kwantum teleportatie voorbeeld enkel om de rechtstreekse aanpak te verklaren.

De kwantumberekening kent drie invoerqubits $|\psi\rangle$ en $|\beta_{00}\rangle$. De eerste qubit $|\psi\rangle = a|0\rangle + b|1\rangle$ mag willekeurig gekozen worden. De tweede en derde qubit $|\beta_{00}\rangle = |\phi_1\rangle \otimes |\phi_2\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$ heet men een EPR-paar (zie sectie 2.2.4). De volledige begintoestand ziet er nu als volgt uit.

$$|\psi_0\rangle = |\psi\rangle \otimes |\beta_{00}\rangle = \begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{a}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{a}{\sqrt{2}} \\ \frac{b}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{b}{\sqrt{2}} \end{pmatrix} \tag{6.1}$$

In de eerste stap van de kwantumberekening wordt de conditionele Pauli-X (ook conditionele negatie of 'CNOT' genoemd) toegepast op de tweede qubit met de eerste qubit als conditie. De matrixvoorstelling van vaak voorkomende conditionele kwantumoperatoren bevindt zich in tabel 3.3. De kwantumoperator O_1 ziet er als volgt uit.

$$O_1 = CNOT \otimes I = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{6.2}$$

Daarna wordt de Hadamard operator, vermeld in tabel 3.2, toegepast op de eerste qubit. De kwantumoperator O_2 voor deze bewerking ziet er als volgt uit.

$$O_2 = H \otimes I \otimes I = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} \end{pmatrix} \tag{6.3}$$

In de voorlaatste stap wordt de conditionele Pauli-X operator toegepast op de laatste qubit met de tweede qubit als conditie. We kunnen deze operator O_3 als volgt formuleren.

$$O_3 = I \otimes CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{6.4}$$

In de laatste stap wordt de conditionele Pauli-Z operator toegepast op de laatste qubit, met de eerste qubit als conditie. Achter het berekenen van de matrixvoorstelling voor deze operator, schuilt een belangrijk optimalisatiemechanisme. Daarom is de verklaring van de berekeningswijze uitgesteld tot in sectie 6.1.3. We nemen voorlopig aan dat de matrix O_4 de gewenste kwantumberekening teweeg brengt.

$$O_4 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \tag{6.5}$$

De operator O, die de volledige kwantumberekening omvat, kan nu berekend worden door de operatoren van alle tussenstappen met elkaar te vermenigvuldigen. Dit staat weergegeven in formule (6.6). De volgorde van bewerkingen moet gerespecteerd worden daar de vermenigvuldiging van matrices niet commutatief is. Het concrete resultaat van deze berekening staat in formule (6.7).

$$O = O_4 \times O_3 \times O_2 \times O_1 \tag{6.6}$$

$$O = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 \end{pmatrix} \tag{6.7}$$

We bepalen nu de eindtoestand van het kwantumcircuit door de operator O toe te passen op de begintoestand $|\psi_0\rangle$. Dit geeft ons de volgende merkwaardige eindtoestand $|\psi_1\rangle$.

$$|\psi_1\rangle = O|\psi_0\rangle = \begin{pmatrix} a/2 \\ b/2 \\ a/2 \\ b/2 \\ a/2 \\ b/2 \\ a/2 \\ b/2 \end{pmatrix} \tag{6.8}$$

Een partiële meting[1] van de eerste en tweede qubit van $|\psi_1\rangle$ zal steeds de enkelvoudige qubit $a|0\rangle + b|1\rangle$ opleveren. Onder teleportatie wordt nu begrepen dat de amplitudes a en b van de eerste qubit $|\psi\rangle$ geteleporteerd zijn naar de derde qubit.

Implementatie van deze aanpak

Een letterlijke implementatie van deze werkwijze heeft te kampen met zware reken- en geheugenproblemen.

1. De matrixelementen van een kwantumoperator zijn complexe getallen. Een matrixvermenigvuldiging of tensorproduct is bijgevolg op laag niveau een reeks van kostbare 'floating-point' bewerkingen. Deze bewerkingen keren heel vaak terug:
   (a) Het bepalen van de basisoperatoren O_1, O_2, O_3, O_4;
   (b) het samenstellen van de basisoperatoren tot O en

[1] Formele definitie van meting staat in sectie 2.2.3. De implementatie ervan staat in sectie 6.2.

(c) het toepassen van de operator O op de invoerqubits $|\psi_0\rangle$.

2. Vanaf een bepaald aantal qubits n zal het geheugengebruik van de kwantumoperator O exhuberant groot zijn door het exponentiële verband tussen n en de dimensie $2^n \times 2^n$ van de matrix O. In de praktijk zou een kwantumberekening met $n > 15$ qubits niet simuleerbaar zijn op een moderne kantoorcomputer anno 2005.

Bovenstaande opmerkingen zijn in strijd met het derde objectief. Dit objectief geeft aan dat er zoveel mogelijk gesnoeid moet worden in de exponentiële groei van de rekentijd en het geheugengebruik. Deze rechtstreekse aanpak kent geen enkele vorm van optimalisatie.

Er werd uiteindelijk toch gekozen om deze aanpak te integreren in de simulator voor diverse redenen. Het expliciet genereren van de operatoren is een educatief en experimenteel middel om bepaalde kwantumberekeningen te kunnen doorgronden. Daarnaast kan deze methode gebruikt worden als maatstaf om optimalisaties te beoordelen.

De functie `qc-apply-straight` past `-operator-` toe op `_qureg_`. Dit is een duidelijk voorbeeld van hoe de klassieke en kwantum abstractielaag elkaar aanvullen, zoals beschreven door het zevende objectief. De kwantumberekening wordt omgezet in een matrixvermenigvuldiging. De resulterende matrix wordt met behulp van de functie `transform-qureg` terug omgezet naar een kwantumregister.

```
(defun qc-apply-straight (_qureg_ -operator-)
  "perform a qubit transformation"
  (transform-qureg
    (multiply-matrices (qop-matrix -operator-)
                       (qureg-matrix _qureg_))))
```

Het is mogelijk om een opeenvolging van kwantumoperatoren te specificeren met behulp van de macro `qc-apply`. In onderstaand voorbeeld passen we achtereenvolgens de Hadamard `-h-`, Pauli-X `-x-` en Pauli-Z `-z-` toe op het kwantumregister `_qureg_`.

```
(qc-apply _result_ (-h- -x- -z-))
```

De rechtstreekse implementatie van het kwantum teleportatie voorbeeld staat hieronder weergegeven. We leggen kort uit hoe deze programmacode werkt.

```
(defun perform-teleportation ()
  (let* ((_psi_  (init-qureg
                   ((/ 1 2) (/ (sqrt 3) 2))))
         (_b00_  (init-qureg
                   ((/ 1 (sqrt 2)) 0 0 (/ 1 (sqrt 2)))))
         (_psi0_ (tensor-items _psi_ _b00_))
         (-o1-   (tensor-items -cnot- -i-))
         (-o2-   (tensor-items -h- -i- -i-))
         (-o3-   (tensor-items -i- -cnot-))
         (-o4-   (tensor-n -i- 3)))
    (setf (get-element -o4- (make-entry 5 5)) -1)
    (setf (get-element -o4- (make-entry 7 7)) -1)
    (partial-measure-qureg
     (qc-apply _psi0_ -o1- -o2- -o3- -o4-) '(0 1))))
```

De `init-qureg` laat toe manueel elke amplitude van een kwantumregister te specificeren. In dit geval is `_psi_` $= \frac{1}{2}|0\rangle + \frac{\sqrt{3}}{2}|1\rangle$ en `_b00_` $= |\beta_{00}\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$. De begintoestand `_psi0_` is het tensorproduct van `_psi_` en `_b00_`. Dit laatste gebeurt met de generieke functie `tensor-items` die ook voor kwantumoperatoren kan gebruikt worden. De conditionele Pauli-Z operator is gelijk aan de eenheidsmatrix met uitzondering van 2 elementen, deze werden afzonderlijk gespecificeerd. De functie `qc-apply` past achtereenvolgens de operatoren `-o1-`, `-o2-`, `-o3-` en `-o4-` toe op het kwantumregister `_psi0_`. Het resultaat is een enkelvoudige qubit als gevolg van een partiële meting met `partial-measure-qureg` van de eerste en tweede qubit.

6.1.3 Specifieke optimalisaties

Optimalisatie van de conditionele Pauli-Z kwantumoperator

We hernemen het kwantum teleportatie voorbeeld uit de vorige sectie 6.1.2. In de laatste stap van deze kwantumberekening wordt de Pauli-Z operator toegepast op de derde qubit, met de eerste qubit als conditie. De berekeningswijze van deze matrix werd met opzet uitgesteld tot in deze sectie. Dit komt omdat de structuur van de Pauli-Z operator een belangrijke optimalisatie teweeg brengt.

De elementen van de enkelvoudige Pauli-Z operator zijn, op het element rechtsonder na, dezelfde als de eenheidsmatrix I. Achter deze eigenschap[2] schuilt een belangrijke performantiewinst. Als we de (conditionele) Pauli-Z

[2]De fase en $\pi/8$ operatoren voldoen ook aan deze merkwaardige eigenschap, zie tabel 3.2 op pagina 30.

operator toepassen op een willekeurige n-qubit, wordt een matrixvermenigvuldiging overbodig. We gaan na hoe dit mogelijk is.

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

De informele betekenis van deze kwantumoperator is dat enkel de amplitude b van een willekeurige enkelvoudige qubit $|\psi\rangle = a|0\rangle + b|1\rangle$ zal gewijzigd worden met een factor -1. Met andere woorden, enkel de amplitude geassocieerd met de basistoestand $|1\rangle$ zal gewijzigd worden. Dit idee staat weergegeven in onderstaande matrixvermenigvuldiging.

$$Z|\psi\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} a \\ -b \end{pmatrix} = a|0\rangle - b|1\rangle$$

We keren nu terug naar de laatste berekeningsstap van het kwantum teleportatie voorbeeld uit de vorige sectie. Er wordt in dit voorbeeld gebruik gemaakt van een 3-qubit $|\psi_0\rangle = |\psi\rangle \otimes |\beta_{00}\rangle$. We beschrijven deze toestand voluit als een lineaire combinatie. De waarde van de amplitudes a_i speelt geen rol in onze uiteenzetting.

$$|\psi_0\rangle = a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|100\rangle + a_5|101\rangle + a_6|110\rangle + a_7|111\rangle$$

De bedoeling is om de kwantumoperator Z toe te passen op de derde qubit $|\phi_2\rangle$, als de eerste conditionele qubit $|\psi\rangle$ gezet is. We gaan deze bewerking uitvoeren in twee stappen om duidelijk aan te geven wat er precies gebeurt.

In de eerste stap passen we onconditioneel de operator Z toe op de derde qubit $|\phi_2\rangle$. We weten dat de Pauli-Z operator enkel een effect heeft als de doelqubit gezet is, met andere woorden enkel de amplitudes met de basistoestand $|??1\rangle$ zullen vermenigvuldigd worden met een factor -1. Het resultaat ziet er als volgt uit.

$$|\psi_0\rangle = a_0|000\rangle - a_1|001\rangle + a_2|010\rangle - a_3|011\rangle + a_4|100\rangle - a_5|101\rangle + a_6|110\rangle - a_7|111\rangle$$

In de tweede stap passen we ook de conditie toe. Dit betekent dat de bovenstaande voorwaarde $|??1\rangle$ uitgebreid wordt naar $|1?1\rangle$ omdat de eerste conditionele qubit $|\psi\rangle$ gezet moet zijn. Enkel de basistoestanden $|101\rangle$ en $|111\rangle$ voldoen aan het opgelegde patroon. Het eindresultaat ziet er als volgt uit.

$$|\psi_0\rangle = a_0|000\rangle + a_1|001\rangle + a_2|010\rangle + a_3|011\rangle + a_4|100\rangle - a_5|101\rangle + a_6|110\rangle - a_7|111\rangle$$

We kunnen deze optimalisatie eenvoudig implementeren door gebruik te maken van de `qureg-do` macro die werd geïntroduceerd in sectie 5.3.3. Deze

macro kan efficiënt itereren over de amplitudes van een kwantumregister op basis van een gegeven bitpatroon. In dit geval zou een matrixvermenigvuldiging tussen een 8×8 matrix en een 1×8 matrix gereduceerd worden tot twee eenvoudige vermenigvuldigingen.

Onderstaande functie implementeert deze specifieke optimalisatie.

```
(defun pauli-z (_qureg_ qid c-qid)
  "perform conditional pauli-z operator"
  (let ((_result_ (copy-qureg _qureg_))
        (filter (make-normal-filter `((,qid 1) (,c-qid 1)))))
    (qureg-do (_result_ basis amplitude filter)
      (setf (get-amplitude _result_ basis) (- amplitude)))
    _result_))
```

De parameter `_qureg_` is een willekeurig kwantumregister van grootte n. De Pauli-Z operator wordt toegepast op de enkelvoudige qubit met index `qid` $\in [0, 1, \ldots, 2^n - 1]$. De index van de conditionele qubit `c-qid` bevindt zich ook in dit interval. Uiteraard geldt dat `qid` $\neq$ `c-qid`.

Het is nu eenvoudig om uit dit idee een matrixvoorstelling af te leiden voor de laatste stap uit het kwantum teleportatie voorbeeld. Enkel de amplitudes van de basistoestanden $|101\rangle$ en $|111\rangle$ worden vermenigvuldigd met een factor -1. Dit betekent dat we de eenheidsmatrix I met dimensie 8×8 in beschouwing nemen, maar de diagonaalelementen op positie[3] $(5, 5)$ en $(7, 7)$ vervangen door -1. Het resultaat staat in formule (6.5) op pagina 79.

Optimalisatie van de wisseloperator

De taak van de wisseloperator is om twee enkelvoudige qubits $|x_i\rangle$ en $|x_j\rangle$, die deel uitmaken van n-qubit $|x_0 x_1 \ldots x_{n-1}\rangle$, van plaats te wisselen. Daarbij geldt dat $i \neq j$ en $x_i \in \{0, 1\}$. De wissel kan geïmplementeerd worden door alle amplitudes a_i, a_j met basistoestanden $a_i|x_0 x_1 \ldots x_{n-1}\rangle$ (waarbij $x_i = 0$ en $x_j = 1$) en $a_j|x_0 x_1 \ldots x_{n-1}\rangle$ (waarbij $x_i = 1$ en $x_j = 0$) van plaats te wisselen. Alle overige bits van deze basistoestanden moeten gelijk zijn. De afstand tussen deze twee basistoestanden bedraagt steeds $2^{n-i-1} - 1$.

Voorbeeld 9 *Beschouwen onderstaande 4-qubit* $|\psi_1\rangle = |q_0 q_1 q_2 q_3\rangle$.

$$\begin{aligned} |\psi_1\rangle \;=\; & a_0|0000\rangle + a_1|0001\rangle + a_2|0010\rangle + a_3|0011\rangle + a_4|0100\rangle + a_5|0101\rangle \\ & + a_6|0110\rangle + a_7|0111\rangle + a_8|1000\rangle + a_9|1001\rangle + a_{10}|1010\rangle + a_{11}|1011\rangle \\ & + a_{12}|1100\rangle + a_{13}|1101\rangle + a_{14}|1110\rangle + a_{15}|1111\rangle \end{aligned}$$

[3] De indices starten vanaf 0.

We wisselen qubits $|q_1\rangle$ en $|q_3\rangle$ in $|\psi_1\rangle$. Volgende amplitudes moeten daarvoor gewisseld worden: $a_1|0001\rangle$ met $a_4|0100\rangle$, $a_3|0011\rangle$ met $a_6|0110\rangle$, $a_9|1001\rangle$ met $a_{12}|1100\rangle$ en $a_{11}|1011\rangle$ met $a_{14}|1110\rangle$. Dit geeft ons volgend resultaat.

$$\begin{aligned}|\psi_2\rangle \;=\; & a_0|0000\rangle + a_4|0001\rangle + a_2|0010\rangle + a_6|0011\rangle + a_1|0100\rangle + a_5|0101\rangle \\ & + a_3|0110\rangle + a_7|0111\rangle + a_8|1000\rangle + a_{12}|1001\rangle + a_{10}|1010\rangle + a_{14}|1011\rangle \\ & + a_9|1100\rangle + a_{13}|1101\rangle + a_{11}|1110\rangle + a_{15}|1111\rangle\end{aligned}$$

We implementeren dit idee door gebruik te maken van een filter (zie sectie 5.3.3). Er wordt gefilterd op basistoestanden waarbij `qid1` $= x_i = 0$ en `qid2` $= x_j = 1$ met `qid1` $<$ `qid2`. De variabele `jump` is de afstand tussen twee te wisselen amplitudes.

```
(defun swap (_qureg_ qid1 qid2)
  "perform a swap operation with qid1 < qid2"
  (let* ((_result_ (copy-qureg _qureg_))
         (jump (1- (expt 2 (- (qureg-size _qureg_) qid1 1))))
         (filter (make-normal-filter '((,qid1 0) (,qid2 1)))))
    (qureg-do
      (_result_ basis amplitude filter)
      (setf (get-amplitude _result_ basis)
        (get-amplitude _qureg_ (+ basis jump)))
      (setf (get-amplitude _result_ (+ basis jump)) amplitude))
    _result_))
```

Optimalisatie van Hadamard kwantumoperator

De berekening waarbij de kwantumoperator H wordt toegepast op een enkelvoudige qubit $|q_i\rangle$ met $i \in [0 \ldots n-1]$, die deel uitmaakt van een willekeurige n-qubit $|\psi\rangle = |q_0 q_1 \ldots q_{n-1}\rangle$, kan geoptimaliseerd worden. We beperken ons in deze sectie tot een *informele gedragsbeschrijving* van de Hadamard operator. De veralgemening van dit geval wordt samen met de wiskundige afleiding behandeld in de volgende sectie 6.1.4.

Voorbeeld 10 *We passen H toe op de enkelvoudige qubit $|\psi\rangle = a|0\rangle + b|1\rangle$ en noemen het resultaat $|\psi'\rangle = a'|0\rangle + b'|1\rangle$. De amplitudes a' en b' worden respectievelijk berekend met de formules (6.9) en (6.10).*

$$a' = \frac{a+b}{\sqrt{2}}, b' = \frac{a-b}{\sqrt{2}}$$

De qubit $|\psi'\rangle$ ziet er nu als volgt uit.

$$|\psi'\rangle = \frac{a+b}{\sqrt{2}}|0\rangle + \frac{a-b}{\sqrt{2}}|1\rangle$$

Het algemene gedrag van de kwantumoperator H kan men als volgt beschrijven. Beschouwen we het bitwoord $x_0x_1 \ldots x_{n-1}$ dat bestaat uit $x_i \in \{0, 1\}$. De amplitudes van de basistoestanden met bitpatroon $x_0x_1 \ldots x_{n-1}$ waarbij $x_i = 0$ zien er, na toepassing van de Hadamard operator op qubit $|q_i\rangle$, als volgt uit.

$$\frac{a_{x_0x_1 \ldots x_{n-1}} + a_{x_0x_1 \ldots \neg x_i \ldots x_{n-1}}}{\sqrt{2}} \tag{6.9}$$

Formule (6.10) geeft de waarde van de andere helft amplitudes weer waarvoor geldt dat $x_i = 1$.

$$\frac{a_{x_0x_1 \ldots \neg x_i \ldots x_{n-1}} - a_{x_0x_1 \ldots x_{n-1}}}{\sqrt{2}} \tag{6.10}$$

Voorbeeld 11 *We hernemen het vorige voorbeeld, maar werken nu met een 3-qubit* $|\psi\rangle = a|000\rangle + b|001\rangle + c|010\rangle + d|011\rangle + e|100\rangle + f|101\rangle + g|110\rangle + h|111\rangle$. *We passen H toe op de tweede qubit (*$i = 1$*) en noemen het resultaat* $|\psi'\rangle$. *We passen formule (6.9) toe op de amplitudes* a', b', e', f' *daar deze geassocieerd zijn met de basistoestand* $|x_0 0 x_2\rangle$.

$$a' = \frac{a+c}{\sqrt{2}}, b' = \frac{b+d}{\sqrt{2}}, e' = \frac{e+g}{\sqrt{2}}, f' = \frac{f+h}{\sqrt{2}}$$

De overige amplitudes c', d', g', h' *kunnen nu eenvoudig verkregen worden door de som te vervangen door een verschil.*

$$c' = \frac{a-c}{\sqrt{2}}, d' = \frac{b-d}{\sqrt{2}}, g' = \frac{e-g}{\sqrt{2}}, h' = \frac{f-h}{\sqrt{2}}$$

De onderstaande functie `hadamard` implementeert de optimalisatie. De parameter `qid` stemt overeen met de index i. De variabele `jump` geeft het decimale verschil weer tussen de bitpatronen $x_0 \ldots x_{n-1}$ met $x_i = 0$ en $x_0 \ldots x_{n-1}$ met $x_i = 1$. Het gebruik van de macro `qureg-do` is hier essentieel om er leesbare programmacode op na te houden.

```
(defun hadamard (_qureg_ qid)
  "perform hadamard operator"
  (let ((_result_ (copy-qureg _qureg_))
        (jump (exp2 (- (qureg-size _qureg_) qid 1)))
        (filter1 (make-normal-filter `((,qid 0))))
        (filter2 (make-normal-filter `((,qid 1)))))
    (qureg-do
      (_result_ basis amplitude filter1)
```

```
  (setf (get-amplitude _result_ basis)
    (/ (+ amplitude
          (get-amplitude _qureg_ (+ basis jump)))
       (sqrt 2))))
(qureg-do
  (_result_ basis amplitude filter2)
  (setf (get-amplitude _result_ basis)
    (/ (- (get-amplitude _qureg_ (- basis jump)) amplitude)
       (sqrt 2))))
_result_))
```

6.1.4 Algemene optimalisatie

We veralgemenen het geval met de Hadamard kwantumoperator uit de vorige sectie 6.1.3. Beschouwen we daartoe een willekeurige kwantumoperator V waarvoor geldt dat $VV^\dagger = I$. De bedoeling is om deze operator V toe te passen op een enkelvoudige qubit $|q_i\rangle$ met $i \in [0 \ldots n-1]$ die deel uitmaakt met een willekeurige n-qubit $|\psi\rangle = |q_0 q_1 \ldots q_{n-1}\rangle$.

$$V = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{6.11}$$

Beschouwen we de kwantumoperatoren O_i die de operator V toepassen op de i-de qubit van $|\psi\rangle$. We illustreren de structuur van de matrices O_i op basis van een eenvoudig voorbeeld met een 3-qubit. De matrixvoorstellingen van alle mogelijke gevallen (O_0, O_1 en O_2) staan respectievelijk weergegeven in de formules (6.12), (6.13) en (6.14). De elementen die 0 zijn, worden niet weergegeven om de leesbaarheid te bevorderen.

$$O_0 = V \otimes I \otimes I = \begin{pmatrix} a & \cdots & \cdots & \cdots & b & & & \\ \vdots & a & & & \vdots & b & & \\ \vdots & & a & & \vdots & & b & \\ \vdots & & & a & \vdots & & & b \\ c & \cdots & \cdots & \cdots & d & & & \\ & c & & & & d & & \\ & & c & & & & d & \\ & & & c & & & & d \end{pmatrix} \tag{6.12}$$

$$O_1 = I \otimes V \otimes I = \begin{pmatrix} a & \cdots & b & & & & & \\ \vdots & a & \vdots & b & & & & \\ c & \cdots & d & & & & & \\ & c & & d & & & & \\ & & & & a & & b & \\ & & & & & a & & b \\ & & & & c & & d & \\ & & & & & c & & d \end{pmatrix} \tag{6.13}$$

$$O_2 = I \otimes I \otimes V = \begin{pmatrix} a & b & & & & & & \\ c & d & & & & & & \\ & & a & b & & & & \\ & & c & d & & & & \\ & & & & a & b & & \\ & & & & c & d & & \\ & & & & & & a & b \\ & & & & & & c & d \end{pmatrix} \tag{6.14}$$

Bovenstaande matrices beschikken over twee belangrijke kenmerken. Elke rij van de matrices O_0, O_1 en O_2 bevat precies functionele elementen (a en b, of c en d). Daarnaast is het mogelijk om de *geprojecteerde afstand* d af te leiden uit het aantal qubits n en de index i met behulp van formule (6.15). Onder de geprojecteerde afstand kan men het aantal elementen begrijpen dat de (horizontale of verticale) stippellijn bestrijkt in de matrices O_0 en O_1. Deze stippellijn is niet weergegeven in matrix O_2 omdat de geprojecteerde afstand daar $d = 1$ is.

$$d = 2^{n-i-1} \tag{6.15}$$

Het is nu mogelijk om in $2^n/2$ iteratiestappen de matrixvermenigvuldiging $O_i|\psi\rangle$ berekenen door gebruik te maken van de geprojecteerde afstand d. We gaan na hoe deze optimalisatie precies in zijn werk gaat. Onderstaand vereenvoudigd code-extract implementeert het optimalisatie-algoritme. De variabele `_qureg_` is een willekeurig kwantumregister, `-operator-` is een enkelvoudige kwantumoperator en `qid` stemt overeen met de index i.

```
1:(defun qc-apply-single (_qureg_ -operator- qid)
2:  "apply -operator- on single qubit of _qureg_"
3:  (let* ((size (qureg-size _qureg_))
4:         (_result_ (make-qureg size)))
5:         (d (exp2 (- size qid 1))
```

```
 6:         (asize (amplitude-count _qureg_)))
 7:    (loop for i from 0 below asize by (* 2 d) do
 8:      (loop for j from i below (+ i d) do
 9:        (progn
10:          (setf (get-amplitude _result_ j)
11:            (+ (* (get-amplitude _qureg_ j)
12:                  (get-element -operator- (make-entry 0 0)))
13:               (* (get-amplitude _qureg_ (+ j d))
14:                  (get-element -operator- (make-entry 1 0)))))
15:          (setf (get-amplitude _result_ (+ j d))
16:            (+ (* (get-amplitude _qureg_ j)
17:                  (get-element -operator- (make-entry 0 1)))
18:               (* (get-amplitude _qureg_ (+ j d))
19:                  (get-element -operator- (make-entry 1 1)))))))))
20:    _result_))
```

De werking van dit algoritme is op impliciete wijze gevisualiseerd in de matrix O_1 die staat weergegeven in formule (6.13). We kunnen 4 'vierkanten' met dimensie $d \times d$ situeren binnen deze matrix, waarvan er één staat aangeduid met een stippellijn. De eerste twee vierkanten bevinden zich linksboven en zijn volledig afgescheiden van de overige twee vierkanten die zich rechtsonder bevinden.

De buitenste lus (regel 7) verzorgt in O_1 de overgang van de vierkanten linksboven naar de vierkanten rechtsonder. Dit wordt gerealiseerd door in sprongen van grootte $2d$ over alle amplitudes te itereren.

De binnenste lus (regel 8) itereert over alle vierkanten binnen een bepaald interval. Per iteratiestap worden de bovenste (regels 10-14) én de onderste (regels 15-19) twee elementen van een vierkant toegepast op de corresponderende amplitudes. Om dit te realiseren, wordt steeds de geprojecteerde afstand gebruikt. De essentie van de optimalisatie zit hem in feit dat we twee bewerkingen per iteratiestap kunnen uitvoeren.

Dit algoritme kan eenvoudig uitgebreid worden zodat het met conditionele kwantumpoorten overweg kan. Beschouwen we de index c van de conditionele enkelvoudige qubit $|q_c\rangle$ die deel uitmaakt van $|\psi\rangle$ met $c \neq i$. De binnenste lus moet voorzien worden van een extra conditie die nagaat of de c-de bit van de variabele `j` is gezet.

De volledige interface van de `qc-apply-single` functie ziet er als volgt uit. De optionele index `c-qid` verwijst naar de conditionele qubit.

```
(qc-apply-single _qureg_ -operator- qid &optional c-qid)
```

De macro `qc-apply` laat gemakkelijk toe om een opeenvolging van kwantumberekeningen te specificeren. We illustreren dit op basis van een concreet voorbeeld. De functie `cnot-swap` implementeert de wisseloperator op basis van drie conditionele negaties. Het kwantumcircuit en de theorie over deze bewerking staan in sectie 3.3.1 op pagina 29. De optimalisatie van de wisseloperator staat beschreven in sectie 6.1.3.

```
(defun cnot-swap (_qureg_ qid1 qid2)
  "perform swap with CNOT operators"
  (qc-apply _qureg_
            ((-x- qid2 qid1) (-x- qid1 qid2) (-x- qid2 qid1))))
```

De argumenten `(-x- qid2 qid1)`, `(-x- qid1 qid2)` en `(-x- qid2 qid1)` worden van links naar rechts geëvalueerd door `qc-apply-single`.

6.2 Meting

6.2.1 Concept

Wanneer een kwantumregister geobserveerd wordt om zijn toestand te raadplegen, vervalt deze onherroepelijk in één van zijn discrete basistoestanden. Dit proces heet men het meten van een kwantumregister. Met alle amplitudes a van een kwantumregister is een bepaalde kans $|a|^2$ geassocieerd. Het kwantumregister is dus een impliciete discrete kansfunctie. De volledige formele uiteenzetting van de meting staat in sectie 2.2.3 op pagina 10.

6.2.2 Globale meting

Onder een globale meting wordt begrepen dat alle qubits van een willekeurige n-qubit $|\psi\rangle$ tegelijkertijd worden gemeten. We introduceren eerst de functie `collapse-basis` die het probabilistische gedrag van de meting implementeert. De macro `matrix-do` itereert over alle matrixelementen van het kwantumregister `_qureg_` die een n-qubit voorstelt. Enkel elementen die verschillend zijn van 0, worden in beschouwing genomen. Deze subtiele optimalisatie is mogelijk omdat de amplitude $a = 0$ een geassocieerde kans $p(a) = |0|^2 = 0$ heeft. Zodra het willekeurig gegenereerde getal $0 \leq$ `rand` < 1 kleiner wordt dan de geaccumuleerde probabiliteiten, wordt de iteratie stopgezet. De return-waarde is de decimale waarde $m \in [0 \ldots 2^n - 1]$ van de laatst tegengekomen basistoestand.

```
(defun collapse-basis (_qureg_)
```

```
  "returns a basis for the collapse"
  (let ((acc 0) (rand (random 1.0)))
    (entry-row
      (matrix-do ((qureg-matrix _qureg_) entry element
                  :condition #'(lambda () (> rand acc))
                  :skip-zero t :result entry)
        (incf acc (calc-magnitude element))))))
```

We kunnen nu op eenvoudige wijze een functie opstellen voor de globale meting. Het 'verval in een discrete basistoestand' betekent concreet dat de gemeten basistoestand, de amplitude 1 zal aannemen. De overige amplitudes zijn allemaal 0. Dit idee staat weergegeven in onderstaande functie `global-measure-qureg` die een willekeurig kwantumregister `_qureg_` als parameter neemt en een gemeten kwantumregister retourneert.

```
(defun global-measure-qureg (_qureg_)
  "global measurement of quantum register"
  (let ((basis (collapse-basis _qureg_)))
    (make-qureg (qureg-size _qureg_)
                (standard-init basis))))
```

6.2.3 Partiële meting

Een partiële meting houdt in dat slechts een deelverzameling enkelvoudige qubits $|q_i\rangle$ (met $i \in [0 \ldots n-1]$), die deel uitmaken van een willekeurige n-qubit $|\psi_1\rangle = |q_0 q_1 \ldots q_n\rangle$, worden geobserveerd. Wanneer we slechts n_1 van de n qubits meten, krijgen we na de meting een $(n - n_1)$-qubit $|\psi_2\rangle$ die de genormaliseerde amplitudes erft van $|\psi_1\rangle$. We geven eerst een voorbeeld van hoe de partiële meting concreet in zijn werk gaat, om daarna over te gaan tot de implementatie ervan.

Voorbeeld 12 *Beschouwen we een willekeurige 4-qubit* $|\phi_1\rangle = |p_0 p_1 p_2 p_3\rangle = \sum_{i=0}^{3} a_i |i\rangle$. *We voeren een partiële meting uit op de tweede (*$|p_1\rangle$*) en de vierde (*$|p_3\rangle$*) qubit. Veronderstellen we dat deze meting* $m = 4$ *opleverde, dit betekent dat* $p_1 = 1$ *en* $p_3 = 0$. *De amplitudes* a_4, a_6, a_{12} *en* a_{14} *zijn geassocieerd met de basistoestand* $|?1?0\rangle$.

We definiëren nu een nieuwe 2-qubit $|\phi_2\rangle = \sum_{i=0}^{3} b_i |i\rangle$ *die de toestand na de partiële meting bevat. Er geldt nu, op een normalisatiefactor* $l = \frac{\langle \phi_2 | \phi_2 \rangle}{\sqrt{2}}$ *na, dat* $b_0 = l a_4$, $b_1 = l a_6$, $b_2 = l a_{12}$ *en* $b_3 = l a_{14}$.

De functie `partial-measure-qureg` implementeert de partiële meting op dezelfde wijze zoals hierboven staat beschreven in het voorbeeld. De

parameter `_qureg_` is een willekeurig kwantumregister en `sel-pattern` geeft aan welke qubits gemeten moeten worden. Deze laatste wordt gebruikt om een `selection` structure aan te maken. De details hierover zijn behandeld in sectie 5.3.3 op pagina 68.

De variabele `collapse` bevat het bitwoord van de gemeten basistoestand. Op basis van de gespecificeerde `sel-pattern` en het gemeten bitwoord `collapse`, wordt een `filter` structure aangemaakt. De macro `qureg-do` itereert nu efficiënt over alle relevante amplitudes en kopieert ze in het kwantumregister `_result_`. Bij terminatie van het algoritme, wordt het resultaat genormaliseerd.

```
(defun partial-measure-qureg (_qureg_ sel-pattern)
  "partial measurement of quantum register"
  (let* ((sel (make-selection sel-pattern))
         (size (qureg-size _qureg_))
         (collapse (make-binary (collapse-basis _qureg_) size))
         (filter (make-filter sel collapse))
         (_result_ (make-qureg (- size (selection-length sel))))
         (result-basis 0))
    (qureg-do (_qureg_ basis amplitude filter)
      (setf (get-amplitude _result_ result-basis) amplitude)
      (incf result-basis))
    (normalize-qureg _result_)))
```

6.2.4 Optimalisatie aaneensluitende meting

In de praktijk gebeurt het vaak dat de laatste k aaneensluitende qubits van een willekeurige n-qubit $|\psi\rangle = \sum_{i=0}^{2^n-1} a_i|i\rangle$ worden gemeten. Dergelijke situatie kan geoptimaliseerd worden. We verklaren het verloop van deze optimalisatie voor $n = 3$ en $k = 1$ met behulp van tabel 6.1. De eerste kolom i geeft de decimale waarde van alle basistoestanden uit $|\psi\rangle$ weer. De daarop volgende kolommen geven de corresponderende bitwoorden $x_0x_1x_2$ weer.

De bedoeling is nu om een partiële meting uit te voeren op de laatste k qubits van $|\psi\rangle$. Veronderstellen we dat het (globale) decimale meetresultaat $m = 5$ bedraagt. We lezen uit de tabel af dat in dit geval $x_2 = 1$, dit betekent dat alle amplitudes met binaire basistoestand $|??1\rangle$ worden opgenomen in het resultaat na meting. De overige amplitudes worden geschrapt. De decimale indices i waarbij $x_2 = 1$ worden berekend met het volgende algoritme.

- De decimale waarde van de kleinste basistoestand met $x_2 = 1$ is gelijk aan m mod 2^k. De waarde 2^k is de periode van de binaire basistoestan-

den $|x_k x_{k+1} \ldots x_{n-1}\rangle$. In de tabel is $k = 1$, elk vakje van kolom x_2 telt bijgevolg $2^1 = 2$ items.

- Alle basistoestanden met $x_2 = 1$ zijn steeds op een gelijke afstand van elkaar verwijderd. Dit komt omdat we rechts uitgelijnde, k aaneensluitende qubits meten. Deze afstand stemt overeen met 2^k.
- Bij terminatie worden alle amplitudes na meting genormaliseerd.

i	x_0	x_1	x_2
0	0	0	0
1			1
2	0	1	0
3			1
4	1	0	0
5			1
⋮	⋮	⋮	⋮

Tabel 6.1: basistoestanden voor $n = 3$, $k = 1$

We geven een beknopte toelichting over de implementatie van deze optimalisatie. De variabele `size2` $= k$ bevat het aantal te meten (rechts uitgelijnde) qubits en `size1` $= n-k$. Het globale meetresultaat is `basis` $= m$. We gebruiken de krachtige `loop` macro uit ANSI Common Lisp om tegelijkertijd over de indices `basis` en `result-basis` te itereren. De index `basis` wijst gemeten amplitudes uit `_qureg_` $= |\psi\rangle$ aan, de andere index `result-basis` voegt de amplitudes toe aan het eindresultaat `_result_`. Bij terminatie wordt het resulterende kwantumregister genormaliseerd.

```
(defun aligned-partial-measure (_qureg_ size2)
  "optimized measure of right aligned size2 qubits"
  (let* ((size1 (- (qureg-size _qureg_) size2))
         (basis (collapse-basis _qureg_))
         (jump (exp2 size2))
         (start (mod basis (exp2 size2)))
         (_result_ (make-qureg size1)))
    (loop for basis from start below (amplitude-count _qureg_) by jump
          for result-basis from 0 below (amplitude-count _result_) do
      (setf (get-amplitude _result_ result-basis)
        (get-amplitude _qureg_ basis)))
    (normalize-qureg _result_)))
```

Hoofdstuk 7

Toepassingen van kwantumalgoritmes

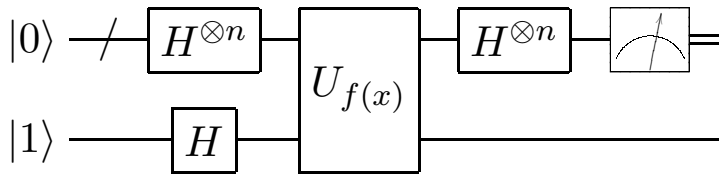
Een kwantumcomputer kan bepaalde berekeningsproblemen efficiënter oplossen dan een klassieke computer. Deze spectaculaire ontdekking wordt in dit laatste hoofdstuk rijkelijk geïllustreerd met populaire kwantumalgoritmes. We duiden steeds nauwgezet aan waar het performantievoordeel zich situeert. Elke uiteenzetting wordt gevolgd door een implementatie.

7.1 Algoritme van Deutsch-Jozsa

Het populaire Deutsch-Jozsa algoritme [DJ92] kan *het probleem van Deutsch* efficiënt oplossen op een kwantumcomputer. Dit probleem kan als volgt beschreven worden. Beschouwen we een functie $f : \{0, 1, \ldots, 2^{n-1}\} \rightarrow \{0, 1\}$. Het algoritme van Deutsch-Jozsa gaat na of de functie $f(x)$ *constant* of *gebalanceerd* is.

We zeggen dat de functie $f(x)$ *constant* is als deze voor alle elementen uit het domein steeds dezelfde waarde (0 of 1) retourneert. Wanneer deze functie voor de helft van het domein een 0 geeft en voor de andere helft een 1, spreekt men over een gebalanceerde functie.

Het beste klassieke algoritme moet de functie $f(x)$ minimaal $2^n/2 + 1$ keer evalueren met een verschillend element uit het domein om na te gaan of de functie constant of gebalanceerd is. Het algoritme van Deutsch-Jozsa kan dit probleem efficiënt oplossen in slechts een enkele functie-evaluatie, steunend op kwantumparallellisme. Het kwantumcircuit van dit algoritme staat voorgesteld in figuur 7.1.



Figuur 7.1: algoritme van Deutsch-Jozsa

7.1.1 Wiskundige uiteenzetting

Het circuit krijgt $n+1$ qubits als invoer waarvan de eerste n qubits voorbereid zijn in de toestand $|0\rangle = 1|0\rangle + 0|1\rangle$ en de laatste qubit in de toestand $|1\rangle = 0|0\rangle + 1|1\rangle$. Formule 7.1 geeft de begintoestand van het algoritme weer.

$$|\psi_0\rangle = |0\rangle^{\otimes n}|1\rangle \tag{7.1}$$

In de eerste stap van het algoritme wordt de Hadamard kwantumoperator toegepast op de eerste n qubits en op de laatste qubit om een superpositie te creëren. Voor de eerste n qubits krijgen we $H^{\otimes n}|0\rangle|0\rangle \ldots |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$ en voor de laatste qubit $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. De volledige begintoestand staat voorgesteld in formule (7.2).

$$|\psi_1\rangle = \frac{1}{2^{n/2}} \left(\sum_{x=0}^{2^n-1} |x\rangle \right) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{7.2}$$

De functie f, waarvan we nagaan of deze constant of gebalanceerd is, dient beschreven te worden door middel van een kwantumoperator U_f. Deze operator neemt als invoer de eerste n qubits en schrijft het resultaat (0 of 1) weg in de laatste qubit $U_f : |x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$. Zoals formule (7.3) aangeeft, *interfereert* de kwantumoperator U_f met alle superposities van $|\psi_1\rangle$. Het performantievoordeel zit hem in feit dat de kwantumoperator U_f in een enkele tijdseenheid wordt toepast op alle 2^n superpositietermen.

$$|\psi_2\rangle = \frac{1}{2^{n/2}} \left(\sum_{x=0}^{2^n-1} (-1)^{f(x)}|x\rangle \right) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{7.3}$$

In de laatste stap voor de meting wordt de Hadamard operator nogmaals toegepast op de eerst n qubits. We construeren eerst een algemene formule voor het toepassen van de Hadamard operator op een willekeurige enkelvoudige qubit $|0\rangle$ of $|1\rangle$.

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}((-1)^{0\cdot 0}|0\rangle + (-1)^{0\cdot 1}|1\rangle) \tag{7.4}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}((-1)^{1\cdot 0}|0\rangle + (-1)^{1\cdot 1}|1\rangle) \tag{7.5}$$

Formule (7.4) en (7.5) kunnen we veralgemenen tot formule (7.6).

$$H|x\rangle = \frac{1}{\sqrt{2}}\sum_{y=0}^{1}(-1)^{x\cdot y}|y\rangle \tag{7.6}$$

Gegeven de algemene formule (7.6) voor het toepassen van de Hadamard operator op een enkelvoudig qubit, construeren we de algemene formule voor n willekeurige qubits $\{x_1, x_2, \ldots, x_n\}$. Merk het onderscheid op tussen de decimale representatie en het binair equivalent $|y\rangle = |y_0 y_1 \ldots y_n\rangle$. We definiëren daartoe het inwendig product modulo 2 als volgt $x\cdot y = x_1y_1 \oplus x_2y_2 \oplus \ldots \oplus x_ny_n$.

$$\begin{aligned}
&H|x_1\rangle \otimes H|x_2\rangle \otimes \ldots \otimes H|x_n\rangle \\
&= \left(\frac{1}{\sqrt{2}}\sum_{y_1=0}^{1}(-1)^{x_1\cdot y_1}|y_1\rangle\right) \otimes \left(\frac{1}{\sqrt{2}}\sum_{y_2=0}^{1}(-1)^{x_2\cdot y_2}|y_2\rangle\right) \otimes \ldots \\
&\quad \otimes \left(\frac{1}{\sqrt{2}}\sum_{y_n=0}^{1}(-1)^{x_n\cdot y_n}|y_n\rangle\right) \\
&= \frac{1}{2^{n/2}}\sum_{y_1 y_2 \ldots y_n}(-1)^{x_1\cdot y_1}(-1)^{x_2\cdot y_2}\ldots(-1)^{x_n\cdot y_n}|y_1 y_2 \ldots y_n\rangle \\
&= \frac{1}{2^{n/2}}\sum_{y=0}^{2^n-1}(-1)^{x\cdot y}|y\rangle
\end{aligned} \tag{7.7}$$

We passen nu formule (7.7) toe op de eerste n qubits uit formule (7.3).

$$\begin{aligned}
|\psi_3\rangle &= \frac{1}{2^{n/2}}\left(\sum_{x=0}^{2^n-1}(-1)^{f(x)}H^{\otimes n}|x\rangle\right) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\
&= \frac{1}{2^{n/2}}\left(\sum_{x=0}^{2^n-1}(-1)^{f(x)}\frac{1}{2^{n/2}}\sum_{y=0}^{2^n-1}(-1)^{x\cdot y}|y\rangle\right) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\
&= \frac{1}{2^n}\left(\sum_{x=0}^{2^n-1}\sum_{y=0}^{2^n-1}(-1)^{x\cdot y+f(x)}|y\rangle\right) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)
\end{aligned} \tag{7.8}$$

De meetwaarde m van de eerste n qubits uit $|\psi_3\rangle$ geeft aan of de functie constant ($m = 0$) of gebalanceerd ($m \neq 0$) is. In de volgende paragrafen tonen we aan vanwaar deze voorwaardes afkomstig zijn.

Veronderstellen we dat $f(x)$ constant is. Dit betekent dat $f(x) = 0$ of dat $f(x) = 1$. We proberen na te gaan wat de amplitude van de basistoestand $|0\rangle$ is, we zeggen dus dat $y = 0$. Uit formule (7.8) kan de amplitude van de basistoestand $|0\rangle$ afgeleid worden. Daar $f(x)$ constant is, zal de som in (7.9) ofwel 1 ofwel -1 zijn. Bijgevolg zullen andere amplitudes 0 zijn. Een meting van de eerste n qubits zal dus altijd $m = 0$ aangeven.

$$\frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \tag{7.9}$$

In het andere geval, waar de functie $f(x)$ gebalanceerd is, ziet de amplitude van de basistoestand $|0\rangle$ er anders uit. We nemen hier opnieuw aan dat $y = 0$ en berekenen de som van formule (7.9). Daar $f(x)$ in de helft van de gevallen 0 is en in de andere helft 1, zullen alle positieve en negatieve termen elkaar opheffen. Met andere woorden, een meting van de eerste n qubits zal steeds $m \neq 0$ aangeven.

7.1.2 Implementatie

De functie `balanced-fn` implementeert de unitaire operator U_f met gebalanceerde functie $f(x) = 0$ als x even is en $f(x) = 1$ als x oneven is.

```
(defun balanced-fn (_qureg_)
  "implements |x,y> -> |x,y+f(x)> with
   f(x)=0 if x is even, f(x)=1 if x is odd"
  (let ((_result_ (copy-qureg _qureg_)))
    (loop for basis from 2 below (amplitude-count _qureg_) by 4 do
      (setf (get-amplitude _result_ basis)
              (get-amplitude _qureg_ (1+ basis))
            (get-amplitude _result_ (1+ basis))
              (get-amplitude _qureg_ basis)))
    _result_))
```

Met behulp van het Deutsch-Jozsa algoritme, verifiëren we nu of de functie $f(x)$ van `balanced-fn` $= U_f$ constant of gebalanceerd is. De hulpfunctie `qc-apply-range` past de Hadamard kwantumoperator `-h-` toe op de eerste `n` qubits van het kwantumregister `_psi_`.

```
(defun deutsch-jozsa (n unitary-fn)
  "returns T if unitary-fn is constant"
  (let* ((_phi1_ (make-qureg n (hadamard-init)))
         (_phi2_ (qc-apply-straight
```

```
                (make-qureg 1 (standard-init 1)) -h-))
        (_psi_  (funcall unitary-fn
                (tensor-items _phi1_ _phi2_)))
        (m      (collapse-basis
                (qc-apply-range _psi_ -h- 0 (1- n)))))
  (or (= m 0) (= m 1))))
```

De evaluatie van `(deutsch-jozsa 5 #'balanced-fn)` geeft ons `nil`. Een voorbeeld van een U_f operator die constant is, is de eenheidsoperator I. De evaluatie van `(deutsch-jozsa 5 #'identity)` geeft ons in dat geval `t`.

7.2 Kwantum Fourier transformatie

Vooraleer we kunnen overgaan tot het algoritme van Shör, concentreren we ons eerst op de belangrijkste tussenstap van dit algoritme: de kwantum Fourier transformatie. We maken gebruik van de exponentiële performantiewinst die kwantumparallellisme met zich meebrengt om de kwantum Fourier transformatie te berekenen. Het kwantumcircuit voor deze berekening is gebaseerd op de productformule 7.12 die onafhankelijk werden ontdekt door onder andere [CEMM98] en [GN96].

7.2.1 Beschrijving van kwantum Fourier transformatie

De gebruikelijke wiskundige definitie (7.10) van de discrete Fourier transformatie neemt $N = 2^n$ complexe getallen $x_0, x_1, \ldots, x_{N-1}$ als invoer. Het resultaat bevat de getransformeerde complexe getallen $y_0, y_1, \ldots, y_{N-1}$. Deze bewerking kan efficiënt berekend worden op een klassieke computer in $O(n \log n)$ met behulp van het *Fast Fourier Transform* algoritme [CT65].

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k/N} \tag{7.10}$$

De kwantum Fourier transformatie (afgekort met QFT) is precies dezelfde als de klassieke versie. De transformatie werkt in dit geval in op een superpositie van de basistoestanden $|0\rangle, \ldots, |N-1\rangle$. De kwantumversie (7.11) van formule (7.10) is zeer gelijkend.

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k/N} |k\rangle \tag{7.11}$$

Afleiding van de productvoorstelling

We stellen nu een kwantumcircuit op die formule 7.11 implementeert. Een herformulering van bovenstaande definitie naar de *productvoorstelling* (7.12) is daarvoor noodzakelijk. Onderstaande paragrafen beschrijven het verloop van deze omzetting.

We herschrijven eerst de decimale basistoestand $|k\rangle$ in een binaire vorm $|k_1 \dots k_n\rangle$ waarvoor geldt dat $k = k_1 2^{n-1} + k_2 2^{n-2} + \dots + k_n 2^0$. Voor elke bit k_l introduceren we een corresponderende sommatie $\sum_{k_l=0}^{1}$. De factor $k/2^n$ uit de exponent van e wordt ook omgezet naar een binaire vorm. Daarvoor wordt de gelijkheid $k/2^n = \sum_{l=1}^{n} k_l 2^{-l}$ gebruikt. Dit laatste noemt men de *binaire breuken*.

$$\begin{aligned} |j\rangle &= \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi i j k/2^n} |k\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} e^{2\pi i j \left(\sum_{l=1}^{n} k_l 2^{-l}\right)} |k_1 \dots k_n\rangle \end{aligned}$$

We schrijven de sommatie $\sum_{l=1}^{n} k_l 2^{-l}$ uit de exponent van e als een tensorproduct $\otimes$ over de verschillende binaire basistoestanden $|k_l\rangle$.

$$|j\rangle = \frac{1}{\sqrt{2^n}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \bigotimes_{l=1}^{n} e^{2\pi i j k_l 2^{-l}} |k_l\rangle$$

Voorgaande herformulering brengt een mooie vereenvoudiging met zich mee. In plaats van de sommaties $\sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1}$ uit te schrijven, gebruiken we nu het tensorproduct van de binaire basistoestanden $|k_l\rangle$. De basistoestand $|k_l\rangle$ is vergezeld van een sommatie die de superpositie tussen de basistoestanden $|0\rangle$ en $|1\rangle$ voorstelt.

$$\begin{aligned} |j\rangle &= \frac{1}{\sqrt{2^n}} \bigotimes_{l=1}^{n} \left[\sum_{k_l=0}^{1} e^{2\pi i j k_l 2^{-l}} |k_l\rangle \right] \\ &= \frac{1}{\sqrt{2^n}} \bigotimes_{l=1}^{n} \left[|0\rangle + e^{2\pi i j 2^{-l}} |1\rangle \right] \end{aligned}$$

We schrijven nu het tensorproduct voluit en zetten daarbij de decimale factor $j/2^l$ om in binaire vorm. Beschouwen we $j = j_1 \dots j_n$ met $j =$

$j_1 2^{n-1} + j_2 2^{n-2} + \ldots + j_n 2^0$. We gebruiken opnieuw binaire breuken om $j/2^l = \sum_{i=l}^{n} j_i 2^{-i}$ voor te stellen. De binaire breuken worden voorgesteld als $0.j_l \ldots j_n = \sum_{i=l}^{n} j_i 2^{-i}$. Dit alles geeft ons onderstaande productvoorstelling (7.12) van de gewone kwantum Fourier definitie (7.11).

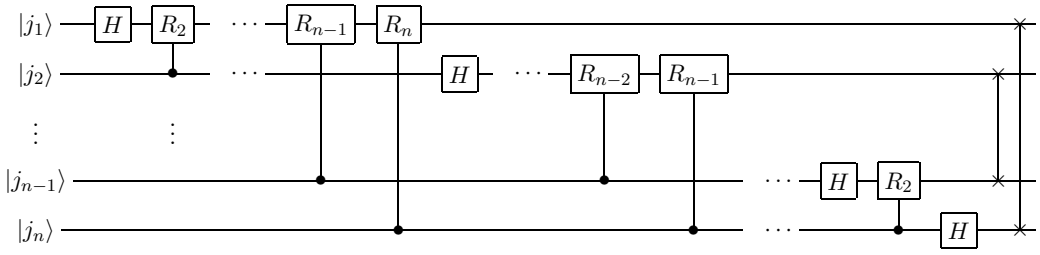
$$|j\rangle = \frac{(|0\rangle + e^{2\pi i 0.j_n}|1\rangle)\,(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle)\cdots(|0\rangle + e^{2\pi i 0.j_1 j_2 \cdots j_n}|1\rangle)}{\sqrt{2^n}} \quad (7.12)$$

Algebraïsche beschrijving van QFT-circuit

De productvoorstelling (7.12) geeft aanleiding tot onderstaand kwantumcircuit. De conditionele operator R_k heeft volgende matrixvoorstelling.

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix} \quad (7.13)$$

In sectie 6.1.3 staat een specifieke optimalisatie voor het toepassen van de conditionele R_k poort op de qubit $|\psi\rangle = a|0\rangle + b|1\rangle$ met de qubit $|\phi\rangle$ als conditie waarbij $\phi \in \{0, 1\}$. We kunnen dit principe hier als volgt algebraïsch begrijpen. We vermenigvuldigen de amplitude b van $|\psi\rangle$ met de factor $r_k = e^{2\pi i \cdot \phi/2^k}$. We zullen dit principe in de komende uiteenzetting regelmatig gebruiken.



Figuur 7.2: kwantum Fourier algoritme

Nu volgt een algebraïsche beschrijving van het verloop van deze kwantumberekening. We zullen zien dat dit hetzelfde resultaat geeft als de productvoorstelling (7.12).

Beschouwen we een willekeurige n-qubit $|j_1 \ldots j_n\rangle$ in binaire vorm. In de eerste stap wordt de Hadamard operator toegepast op $|j_1\rangle$. Dit staat weergegeven in formule (7.14). Het gebruik van de exponentiële functie bij de basistoestand $|1\rangle$ lijkt vreemd, maar merk op dat $e^{2\pi i 0.j_1} = e^{2\pi j_1/2} = -1$ als $j_1 = 1$, en 1 in het andere geval. Deze manier van werken zal de formules in de volgende stappen leesbaarder maken.

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.j_1}|1\rangle\right)|j_2 \ldots j_n\rangle \tag{7.14}$$

In deze eerste stap is $r_2 = e^{2\pi i p_2/4}$. Wanneer de conditie $p_2 = 0$, wordt de factor $r_2 = 1$ en blijft de amplitude van de basistoestand $|1\rangle$ ongewijzigd. In het andere geval, waar $p_2 = 1$, wordt de factor $r_2 = e^{\pi i/2}$ toegepast op de basistoestand $|1\rangle$ zoals operator R_2 voorschrijft.

De vermenigvuldiging van r_2 met de factor $e^{2\pi i 0.j_1}$ van basistoestand $|1\rangle$ uit formule (7.14) ziet er als volgt uit.

$$\begin{aligned} r_2 \cdot e^{2\pi i 0.j_1} &= e^{2\pi i j_2/4} \cdot e^{2\pi i j_1/2} \\ &= e^{2\pi i j_2/4 + 2\pi i j_1/2} \\ &= e^{2\pi i (j_2/4 + j_1/2)} \\ &= e^{2\pi i 0.j_2 j_1} \end{aligned}$$

De toestand na toepassing van de R_2 operator ziet er nu als volgt uit.

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.j_1 j_2}|1\rangle\right)|j_2 \ldots j_n\rangle \tag{7.15}$$

Het toepassen van de conditionele $R_3, R_4, \ldots, R_n$ kwantumoperatoren op $|j_1\rangle$ geeft volgend resultaat.

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle\right)|j_2 \ldots j_n\rangle \tag{7.16}$$

We herhalen nu bovenstaande procedure voor een volgende qubit. Deze keer wordt de Hadamard kwantumoperator toegepast op $|j_2\rangle$ met achtereenvolgens de conditionele $R_3, R_4, \ldots, R_{n-1}$. Dit geeft volgend resultaat.

$$\frac{1}{2}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_2 \ldots j_n}|1\rangle\right)|j_3 \ldots j_n\rangle \tag{7.17}$$

Wanneer we het kwantumcircuit vervolledigen voor alle qubits, krijgen we onderstaande formule. Merk op dat de verschillende factoren worden gecombineerd door middel van een tensorproduct. De volgorde van bewerkingen is dus van belang, zie bijlage A.1.5.

$$\frac{1}{\sqrt{2^n}}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_2 \ldots j_n}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i 0.j_n}|1\rangle\right) \tag{7.18}$$

De verschillende factoren van formule (7.18) stemmen overeen met de factoren uit de productvoorstelling (7.12). De swap-operaties op het einde van het kwantumcircuit zorgen ervoor dat de volgorde van de factoren wordt omgekeerd.

7.2.2 Implementatie

We gebuiken de particuliere optimalisatietechnieken uit sectie 6.1.3 om de R_k en H kwantumoperatoren toe te passen. De implementatie van de geoptimaliseerde R_k operator staat hieronder weergegeven. Merk op dat deze code op hetzelfde principe gebaseerd is als de optimalisatie van de Pauli-Z operator uit sectie 6.1.3. De functie `complex` construeert een complex getal $a+bi$ met $\cos(2\pi/2^k)$ als reële gedeelte a en $\sin(2\pi/2^k)$ als imaginair gedeelte b.

```
(defun controlled-r (_qureg_ k qid c-qid)
  "perform conditional r operator"
  (let* ((e (/ (* 2 pi) (exp2 k)))
         (factor (complex (cos e) (sin e)))
         (_result_ (copy-qureg _qureg_))
         (filter (make-normal-filter `((,qid 1) (,c-qid 1)))))
    (qureg-do (_result_ basis amplitude filter)
      (setf (get-amplitude _result_ basis)
        (* amplitude factor)))
    _result_))
```

Het verloop van de kwantumberekening wordt beschreven met behulp van de krachtige `loop`-macro die deel uitmaakt van ANSI Common Lisp. De geneste lussen berekenen de indices die aangegeven op welke qubits de verschillende kwantumoperatoren toegepast worden. We maken gebruiken van de specifieke optimalisaties zoals `hadamard` en `swap` uit sectie 6.1.3.

```
(defun quantum-fourier (_qureg_)
  "apply quantum fourier via parallel network"
  (let ((size (qureg-size _qureg_))
        (_result_ (copy-qureg _qureg_)))
    (loop for qid from 0 below size do
      (setf _result_ (hadamard _result_ qid))
      (loop for c-qid from (1+ qid) below size
            for k from 2 do
        (setf _result_ (controlled-r _result_ k qid c-qid))))
      (loop for i from 0 below (/ size 2)
            for j from (1- size) above (/ size 2) do
        (setf _result_ (swap _result_ i j)))
    _result_))
```

7.3 Algoritme van Shor

Het algoritme van Shor is het paradepaardje van de kwantumberekeningen. Het werd in 1994 ontwikkeld door Peter Shor die daarmee de Nevanlinna-prijs voor theoretische informatica won. Met dit algoritme kan in polynomiale rekentijd een priemfactorisatie berekend worden. Dit is een messteek in de rug van Rivest, Shamir en Adleman die in 1978 het RSA cryptosysteem [RSA78] ontwikkelden. De veiligheid van RSA is gebaseerd op de onbewezen aanname dat een priemfactorisatie momenteel niet efficiënt kan worden berekend op een klassieke computer [Lan01]. Zie bijlage B.

Het algoritme van Shor trok de bijzondere aandacht van IBM, die momenteel investeert in onderzoek naar hardwarematige implementaties van dit algoritme. Dergelijke realisatie zou een enorme impact hebben op het informaticalandschap, alsook op maatschappelijk vlak. Immers, alle moderne beveiligingen voor computernetwerken zijn op RSA geïnspireerd.

7.3.1 Beschrijving algoritme van Shor

We beschrijven op een hoog niveau de vijf stappen van het algoritme van Shor, waarvan enkel stap (3) een beroep doet op kwantumberekeningen. In de volgende subsecties worden alle stappen in detail behandeld. De invoer van het algoritme is $N \in \mathbb{N} : N > 1$.

1. De eerste stap controleert of een priemfactorisatie nodig is.
   (a) Wanneer N even is, stopt het algoritme en worden de priemfactoren 1 en 2 geretourneerd. Dit komt omdat alle priemfactoren oneven zijn, met uitzondering van het getal 2.
   (b) Ga na of N een priemgetal is. Wanneer dit zo is, wordt het algoritme gestopt.
   (c) Ga na of er een $a \in \mathbb{N} : a \geq 1$ en een $b \in \mathbb{N} : b \geq 2$ bestaat waarvoor geldt dat $N = a^b$.
2. Genereer een willekeurig getal $x \in \{1, 2, 3, \ldots, N-1\}$ dat relatief priem is met N. Dit betekent dat $\text{ggd}(x, N) = 1$. Op die manier weten we zeker dat x en N geen gemeenschappelijke factoren bezitten.
3. Bereken, met behulp van een kwantumberekening, de periode r van onderstaande discrete functie die een modulaire exponentiatie definieert.

$$f_{x,N}(k) = x^k \bmod N \tag{7.19}$$

$$f_{x,N}(k+r) = f_{x,N}(k) \tag{7.20}$$

Dit is equivalent met het zoeken van de orde r van x mod N. De orde r is het kleinst mogelijk positief natuurlijk getal waarvoor geldt dat:

$$x^r = 1(\mathrm{mod} N) \tag{7.21}$$

4. Zolang de gevonden periode r niet even is en $x^{r/2} + 1 \neq 0 \bmod N$, moeten de stappen (2) en (3) herhaald worden.

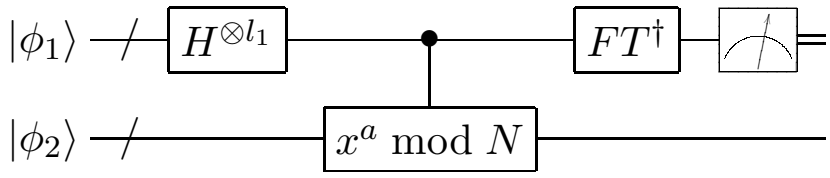
5. De priemfactoren $N = pq$ kunnen nu met onderstaande formules berekend worden.

$$p = \mathrm{ggd}(x^{r/2} - 1, N) \tag{7.22}$$

$$q = \mathrm{ggd}(x^{r/2} + 1, N) \tag{7.23}$$

7.3.2 Stap 3: berekenen van de periode r

We zoomen in deze sectie in op de derde en belangrijkste stap van het algoritme. Het berekenen van de periode r kan efficiënt gebeuren door middel van een kwantumberekening. De beschrijving van deze berekening staat weergegeven in onderstaand kwantumcircuit.



Figuur 7.3: periode r berekenen

We onderscheiden vijf stappen in deze berekening.

1. De Hadamard kwantumoperator wordt toegepast op $|\phi_1\rangle$.

2. De modulaire exponentiatie wordt berekend op basis van $|\phi_1\rangle$, het resultaat wordt weggeschreven in $|\phi_2\rangle$.

3. De tweede qubit $|\phi_2\rangle$ wordt gemeten.

4. De discrete Fourier transformatie wordt toegepast op $|\phi_1\rangle$.

5. Het meetresultaat van $|\phi_1\rangle$ wordt geretourneerd als resultaat van de kwantumberekening.

6. De periode berekenen met behulp van een klassiek algoritme op basis van het meetresultaat uit de vorige stap en het aantal qubits in $|\phi_1\rangle$.

Het aantal bits b van het getal N bedraagt $b = \lceil log_2 N \rceil$. De eerste $2b$-qubit $|\phi_1\rangle$ telt $l_1 = 2^{2b}$ superpositietermen, de tweede b-qubit $|\phi_2\rangle$ telt $l_2 = 2^b$ superpositietermen. De begintoestand $|\psi_0\rangle$ van het kwantumcircuit wordt beschreven door middel van formule (7.24). Merk op dat de basistoestanden steeds decimaal worden voorgesteld.

$$|\psi_0\rangle = |\phi_1\rangle \otimes |\phi_2\rangle = \sum_{a=0}^{l_1-1} \sum_{b=0}^{l_2-1} c_{a,b}|a,b\rangle = |0,0\rangle \tag{7.24}$$

Stap 3.1: superpositie creëren in $|\phi_1\rangle$

We passen de Hadamard kwantumoperator toe op de eerste qubit $|\phi_1\rangle$. De toestand na toepassing van stap (1) noemen we $|\psi_1\rangle$ en ziet er als volgt uit.

$$|\psi_1\rangle = \frac{1}{\sqrt{l_1}} \sum_{a=0}^{l_1-1} |a,0\rangle \tag{7.25}$$

Stap 3.2: modulaire exponentiatie

In de volgende stap (2) wordt de transformatie $|a,0\rangle \mapsto |a, x^a \bmod N\rangle$ toegepast op $|\psi_1\rangle$. Het resultaat $|\psi_2\rangle$ ziet er als volgt uit.

$$|\psi_2\rangle = \frac{1}{\sqrt{l_1}} \sum_{a=0}^{l_1-1} |a, x^a \bmod N\rangle \tag{7.26}$$

Stap 3.3: meting van $|\phi_2\rangle$

We meten nu $|\phi_2\rangle$. De mogelijke meetresultaten m stemmen overeen met $x^m \bmod N$. De kans op meting $p(m) = 1/l_1$ is overal gelijk doordat de Hadamard operator werd toegepast in stap (1).

Veronderstellen we nu dat de meting, het resultaat m opleverde. Dit betekent dat $|\phi_1\rangle$ nu enkel nog over de l_3 basistoestanden $|a_0\rangle, |a_0 + r\rangle, |a_0 + 2r\rangle, \ldots, |a_0 + (l_3 - 1)r\rangle$ beschikt. Het resultaat $|\psi_3\rangle$ van stap (3) staat weergegeven in onderstaande formule.

$$|\psi_3\rangle = \frac{1}{\sqrt{l_3}} \sum_{d=0}^{l_3-1} |a_0 + dr, m\rangle \tag{7.27}$$

Stap 3.4: kwantum Fourier transformatie

In de voorlaatste stap passen we de discrete Fourier transformatie toe op $|\phi_1\rangle$. De algemene formule van de discrete Fourier transformatie voor kwantumberekeningen is equivalent met de klassieke versie. We hernemen formule 7.11, de transformatie gebeurt door middel van onderstaande lineaire operator.

$$|j\rangle \mapsto \frac{1}{\sqrt{n}} \sum_{c=0}^{n-1} e^{2\pi ijc/n} |c\rangle \tag{7.28}$$

We passen de kwantum Fourier transformatie uit formule (7.28) toe op $|\psi_3\rangle$ om de volgende toestand $|\psi_4\rangle$ te berekenen. Daarbij substitueren we j met a_0+dr en n met l_1. Om de leesbaarheid van formule (7.29) te bevorderen, zeggen we dat $\xi = e^{2\pi icr/l_1}$.

$$\begin{aligned} |\psi_4\rangle &= \frac{1}{\sqrt{l_1 l_3}} \sum_{c=0}^{l_1-1} \sum_{d=0}^{l_3-1} e^{2\pi i \frac{c(a_0+dr)}{l_1}} |c, m\rangle \\ &= \sum_{c=0}^{l_1-1} \frac{e^{2\pi i c a_0/l_1}}{\sqrt{l_1 l_3}} \sum_{d=0}^{l_3-1} e^{2\pi i cdr/l_1} |c, m\rangle \\ &= \sum_{c=0}^{l_1-1} \frac{e^{2\pi i c a_0/q}}{\sqrt{l_1 l_3}} \sum_{d=0}^{l_3-1} \xi^d |c, m\rangle \end{aligned} \tag{7.29}$$

Stap 3.5: meting van $|\phi_1\rangle$

We eindigen de kwantumberekening met een meting van $|\phi_1\rangle$. Onderstaande formule geeft de kans weer om basistoestand $|c\rangle$ te meten.

$$\begin{aligned} p(c) &= \left| \frac{e^{2\pi i c a_0/q}}{\sqrt{l_1 l_3}} \sum_{d=0}^{l_3-1} \xi^d \right|^2 \\ &= \frac{1}{l_1 l_3} \left| \sum_{d=0}^{l_3-1} \xi^d \right|^2 \\ &= \frac{1}{l_1 l_3} \left| \frac{1-\xi^{l_3}}{1-\xi} \right|^2 \end{aligned} \tag{7.30}$$

De merkwaardige overgang $\left|e^{2\pi i c a_0/q}\right|^2 = 1$ is mogelijk omdat $\cos(2\pi k) = 1$ en $\sin(2\pi k) = 0$ voor alle $k \in \mathbb{Z}$. Onderstaande formule (7.31) geeft enkele tussenstappen van deze merkwaardige overgang weer.

$$\begin{aligned}\left|e^{2\pi i c a_0/q}\right|^2 &= \left|\sqrt[q]{e^{2\pi i c a_0}}\right|^2 \\ &= \left|\sqrt[q]{\cos(2\pi c a_0) + i\sin(2\pi c a_0)}\right|^2 \\ &= 1 \end{aligned} \tag{7.31}$$

Stap 3.6: periode r berekenen

We proberen nu, op basis van het meetresultaat c en het aantal qubits l_1 van $|\phi_1\rangle$, de periode r te berekenen. We beschrijven daarvoor op informele wijze hoe de kans $p(c)$ evolueert (zie formule 7.30) voor verschillende waardes van cr/l_1. Een gedetailleerde wiskundige en statistische context van deze materie staat in [x]. Beschouwen we het verschil $\delta \in \mathbb{Q}$ tussen de factor cr/l_1 uit $\xi = e^{2\pi i cr/l_1}$ en het meest nabije natuurlijk getal.

$$\delta = \min(\lfloor cr/l_1 \rfloor - cr/l_1, \lceil cr/l_1 \rceil - cr/l_1) \tag{7.32}$$

Als de factor cr/l_1 niet in de nabijheid van een natuurlijk getal ligt, zal $\delta \to \frac{1}{2}$. Als gevolg van *destructieve interferentie*, zal de factor $\frac{1-\xi^{l_3}}{1-\xi} \to 0$ als $l_3 \to \infty$ in formule (7.30). Dit betekent dat de volledig formule $p(c) \to 0$. Met andere woorden, de kans dat cr/l_1 niet in de buurt van een natuurlijk getal ligt, is heel klein.

Wanneer we veronderstellen dat cr/l_1 wel in de nabijheid van een natuurlijk getal ligt, zal $\delta \to 0$. Dit betekent dat voor $s \in \mathbb{N}$ geldt dat:

$$\frac{cr}{l_1} \approx s \tag{7.33}$$

Bijgevolg zal $\xi = e^{2\pi i cr/l_1} \approx 1$ doordat $e^{2\pi i k} = 1$ voor alle $k \in \mathbb{Z}$. In dit geval kunnen we de kansformule (7.30) sterk vereenvoudigen.

$$p(c) \approx \frac{l_3}{l_1 l_3} \approx \frac{1}{l_1} \tag{7.34}$$

De conclusie is dat de kans het grootst is dat cr/l_1 in de nabijheid van een natuurlijk getal ligt. Uit formule (7.33) leiden we nu onderstaande benadering af.

$$\frac{c}{l_1} \approx \frac{s}{r} \tag{7.35}$$

We kunnen *proberen* de periode r af te leiden uit formule (7.35) door de generatie van kettingbreuken (zie bijlage A.3) voor c/l_1. Beschouwen we alle

benaderingen $\left(\widetilde{c/l_1}\right)_i$ met $i \in [0, 1, \ldots, n]$ die steeds dichter bij de werkelijke waarde c/l_1 komen. Voor elke mogelijke benadering $\left(\widetilde{c/l_1}\right)_i$ gaan we na of de potentiële oplossing $r' = \left(\widetilde{l_1}\right)_i$ voldoet aan de voorwaarde uit formule (7.21).

De wiskundige Odlyzko (1996) geeft aan dat het ook interessant is om enkele veelvouden $k \in \mathbb{N}_0$ van de potentiële oplossing r' na te gaan. Deze techniek is mogelijk omdat $\left(\widetilde{kc/kl_1}\right)_i = \left(\widetilde{c/l_1}\right)_i$. Het aantal veelvouden dat gecontroleerd moet worden, met inbegrip van een fout ϵ, staat in formule (7.36).

$$r', 2r', 3r', \ldots, \left\lfloor \log_2(n)^{1+\epsilon} \right\rfloor r' \tag{7.36}$$

7.3.3 Stappen 4 en 5: verifiëren van de periode r

De periode r dient even te zijn, anders is het niet mogelijk om de formules (7.22) en (7.23) te berekenen. De tweede voorwaarde geeft aan dat $x^{r/2} \neq -1(\mathrm{mod}N)$. Als deze voorwaardes niet voldaan zijn, is dit te wijten aan een 'slechte' keuze van de willekeurige variabele x uit stap (2). Het is niet mogelijk om a priori een x te kiezen die tot een 'goede' periode r zal leiden.

Na de controle uit stap (2), kan nu op eenvoudige wijze het eindresultaat berekend worden met formules (7.22) en (7.23).

7.3.4 Uitgewerkt voorbeeld

De concrete werking van het algoritme van Shor wordt geïllustreerd met een uitgewerkt voorbeeld. We zoeken de priemfactoren voor $N = 55$.

Stap 1 en 2: initialisatie van de berekening

We hebben een willekeurige getal $x = 13$ gegenereerd waarvoor geldt dat $\mathrm{ggd}(13, 55) = 1$.

Stap 3: berekenen van de periode r

Het getal $N = 55$ kan beschreven worden met $b = 6$ bits. De eerste qubit $|\phi_1\rangle$ is samengesteld uit 12 enkelvoudige qubits met $l_1 = 2^{12} = 4096$ amplitudes. De tweede qubit $|\phi_2\rangle$ bestaat uit 6 enkelvoudige qubits met $l_2 = 2^6 = 64$ amplitudes. De begintoestand is $|\psi_0\rangle = |\phi_1\rangle \otimes |\phi_2\rangle = |0, 0\rangle$.

Stap 3.1: superpositie creëren in $|\phi_1\rangle$

$$|\psi_1\rangle = \frac{1}{\sqrt{4096}} \left(|0,0\rangle + |1,0\rangle + |2,0\rangle + \ldots + |4095,0\rangle\right)$$

Stap 3.2: modulaire exponentiatie

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{\sqrt{4096}} \left(|0,1\rangle + |1,13\rangle + |2,13^2 \bmod 55\rangle + \ldots + |4095,13^{4095} \bmod 55\rangle\right) \\ &= \frac{1}{\sqrt{4096}} \left(|0,1\rangle + |1,13\rangle + |2,4\rangle + \ldots + |4095,32\rangle\right) \end{aligned}$$

Stap 3.3: meting van $|\phi_2\rangle$

Veronderstellen we dat het meetresultaat $m = 28$ is. Bijgevolg is $a_0 = 9$, $l_3 = 205$ en $r = 20$.

$$|\psi_3\rangle = \frac{1}{\sqrt{205}} \left(|9,28\rangle + |29,28\rangle + |49,28\rangle + \ldots + |4089,28\rangle\right)$$

Stap 3.4: kwantum Fourier transformatie

Voor $\xi = e^{2\pi i \cdot 20c/4096}$ geldt dat:

$$|\psi_4\rangle = \sum_{c=0}^{4095} \frac{e^{2\pi i \cdot 9c/4096}}{\sqrt{4096 \cdot 205}} \sum_{d=0}^{204} \xi^d |c, 28\rangle$$

Stap 3.5: meting van $|\phi_2\rangle$

Veronderstellen we dat de meting het resultaat $c = 1229$ opleverde. De probabiliteit op dit meetresultaat bedraagt $p(1229) = 0,04$.

Stap 3.6: periode r berekenen

Alle tussenresultaten van de continue fracties voor $c/l_1 = 1229/4096$ staan weergegeven in onderstaande tabel.

We ga na welke q_n, of veelvoud daarvan, overeenstemt met de gezochte periode r.

- We slaan $q_0 = 1$ over omdat dit geval triviaal is.

n	0	1	2	3	4
a_n	0	3	3	204	2
p_n	0	1	3	613	1229
q_n	1	3	10	2043	4096

Tabel 7.1: continue fracties voor $c/l_1 = 1229/4096$

- Voor $q_1 = 3$ geldt dat $13^3 \bmod 55 = 52$. We proberen enkele veelvouden.
  - $13^{2 \cdot 3} \bmod 55 = 9$
  - $13^{3 \cdot 3} \bmod 55 = 28$
  - $13^{4 \cdot 3} \bmod 55 = 26$
- We proberen nu $q_2 = 10$. Dit geeft $13^{10} \bmod 55 = 34$. We proberen enkele veelvouden.
  - $13^{2 \cdot 10} \bmod 55 = 1$, dit betekent dat we een periode $r = 2 \cdot 10 = 20$ die in aanmerking komt.

Stap 4: verifiëren van de periode r

De gevonden periode $r = 20$ is even en $13^{20/2} \neq -1(\text{mod}55)$. Beide voorwaardes zijn voldaan.

Stap 5: resultaat berekenen

De priemfactoren p en q voor $N = pq = 55$ zijn:

$$p = \text{ggd}(13^{20/2} - 1, 55) = 11, q = \text{ggd}(13^{20/2} + 1, 55) = 5$$

7.3.5 Implementatie

We bespreken kort het verloop van de functie `shor`. Deze retourneert in het beste geval de beide priemfactoren van `n`. De voorafgaandelijke testen op kwadraten, even getallen en priemgetallen uit stap (1) worden overgeslagen.

- De hulpfunctie `random-base` retourneert een geschikte willekeurige basis `x`.
- De lengte `b` voor de binaire voorstelling van `n` wordt bepaald met `integer-length`.

- De functie `mod-exp` simuleert het gedrag van de conditionele modulaire exponentiatie operator. Daarvoor wordt een beroep gedaan op een efficiënt klassiek algoritme gebaseerd op de techniek van herhaaldelijke kwadraten.
- We gebruiken de specifieke optimalisatie `aligned-partial-measure` uit sectie 6.2.4 om het tweede kwantumregister `_phi2_` te meten.
- De functie `quantum-fourier` is de kwantum Fourier implementatie uit de vorige sectie.
- De periode `r` wordt gezocht door de functie `get-period`. Indien er geen geschikte periode wordt gevonden, wordt `nil` geretourneerd. In dit geval moet het algoritme opnieuw gestart worden.

```
(defun shor (n)
  (let* ((x (random-base n))
         (b (integer-length n))
         (_phi1_ (make-qureg (* 2 b) (hadamard-init)))
         (_phi2_ (make-qureg b))
         (c (collapse-basis
             (quantum-fourier
              (aligned-partial-measure
               (mod-exp _phi1_ _phi2_ x n) (* 2 b)))))
         (r (get-period x n c (amplitude-count _phi1_))))
    (if r (values (get-factor-1 x n r)
                  (get-factor-2 x n r)))))
```

7.4 BB84 Protocol

Het BB84 protocol is een populair versleutelsysteem voor kwantumcomputers die veilige communicatie over afstand garandeert. De initialen 'BB' verwijzen naar de ontwikkelaars Charles H. Bennett en en Gilles Brassard [BB84], de 84 wijst naar het jaar van origine 1984. Dit privaat encryptiesysteem is reeds succesvol hardwarematig geïmplementeerd over een afstand van 120 km. De implementaties zijn nu al commercieel beschikbaar [Mag].

In private encryptiesystemen is er slechts één geheime sleutel waarover de zender en de ontvanger beschikken. Het doel van het algoritme is het uitwisselen (via een onveilig communicatiekanaal) van een geheime sleutel tussen twee partijen. In de vakliteratuur noemt men dit doorgaans *quantum key distribution*.

Men kan bewijzen dat het BB84 protocol veilig is als gevolg van twee belangrijke kwantummechanische kenmerken. Het is niet mogelijk om qubits te kopiëren, dit is de niet-klonen stelling (zie sectie 2.2.4). Daarnaast wordt er creatief ingespeeld op het wisselen tussen de rechthoekige en diagonale berekeningsbasissen (zie sectie 2.2.3). Dit laatste wordt verderop in deze sectie nader verklaard.

Wanneer een qubit wordt verstuurd via een kwantum communicatiekanaal, kunnen we via technieken uit de statistiek en de klassieke foutcorrectie nagaan of de originele qubit onafgeluisterd werd verstuurd. Het is immers mogelijk dat een derde partij de qubit probeert te onderscheppen en een valse qubit in de plaats stuurt.

> "Quantum key distribution is a major paradigm shift in the development of cryptography. The ability to detect eavesdropping on a communications link with absolute certainty is remarkable and conventional and quantum cryptography are a powerful combination in making secure communications a reality."
>
> Burt Kaliski, wetenschapsmanager, RSA Laboratories

7.4.1 Beschrijving van het protocol

De bedoeling van het BB84 protocol is om een private sleutel uit te wisselen tussen twee partijen (Alice en Bob). Het protocol veronderstelt dat er een unidirectioneel kwantumkanaal (van Alice naar Bob) en een bidirectioneel klassiek communicatiekanaal beschikbaar is. De mogelijkheid bestaat dat qubits of bits worden afgeluisterd of veranderd tijdens het versturen over beide communicatiekanalen.

1. Alice kiest $a = (4+\delta)n$ willekeurige *databits* en $b = (4+\delta)n$ willekeurige *basisbits*. Op basis van een a_i en corresponderende b_i, construeert Alice enkelvoudige qubits.

   - Als $b_i = 0$ met $a_i = 0$ of $a_i = 1$, wordt respectievelijk een $|0\rangle$ of $|1\rangle$ qubit geconstrueerd.
   - In het andere geval, waarbij $b_i = 1$, wordt voor $a_i = 0$ of $a_i = 1$ respectievelijk een $|+\rangle$ of $|-\rangle$ qubit geconstrueerd.

2. Alice verstuurt de gegenereerde qubits $q = \left[|q_1\rangle, |q_2\rangle, \ldots, |q_{(4+\delta)n}\rangle\right]$ naar Bob.

3. Bob ontvangt de qubits q'. Wanneer de qubits tijdens het versturen niet werden aangetast, geldt dat $q = q'$. De overige stappen van dit protocol gaan dit achterhalen.

4. Bob kiest ook $b' = (4 + \delta)n$ willekeurige basisbits waarmee hij de ontvangen qubits q' meet. Als $b'_i = 0$, worden de rechthoekige berekeningsbasissen $|0\rangle$ en $|1\rangle$ gebruikt, in het andere geval worden de diagonale berekeningsbasissen $|+\rangle$ en $|-\rangle$ gebruikt. Hij schrijft de resultaten weg in $a' = (4 + \delta)n$ die de gemeten databits bevat.

5. Alice en Bob vergelijken hun basisbits b en b' via een klassiek communicatiekanaal. Beide partijen verwerpen de databits uit a en a' die in de verkeerde berekeningsbasis werden gemeten door Bob. Hoogstwaarschijnlijk blijven er $2n$ of meer databits over. In het andere geval moet het protocol herstart worden.

6. Alice kiest een deelverzameling van n controlebits uit a. Ze stuurt haar geselecteerde controlebits naar Bob via een klassiek communicatiekanaal.

7. Bob controleert of de deelverzameling van n controlebits overeenstemt met zijn databits a'. We noemen de procentuele fout ϵ. Wanneer $\epsilon > \epsilon_{max}$, moet het protocol herstart worden. De oorzaken liggen hiervoor bij het kwantum communicatiekanaal dat:

   - ofwel teveel ruis bevat;
   - ofwel werd afgeluisterd.

   Wanneer het protocol wordt verder gezet, worden alle controlebits verworpen.

8. In de laatste stap passen we *informatie reconciliatie* en een *privacy versterking* toe op de resterende n databits van a en a'.

   - De bedoeling van informatie reconciliatie is om eventuele resterende fouten in a en a' zoveel mogelijk te corrigeren met behulp van klassieke foutcorrectie. Het foutcorrectie algoritme moet uiteraard bestendig zijn tegen afluisteren van het klassieke communicatiekanaal.
   - Ten slotte wordt er een privacy versterking gerealiseerd door een 'exotische' hashfunctie toe te passen op beide databits a en a'.

Het BB84 protocol heeft te kampen met het potentiële "man in het midden"probleem. Veronderstel dat er een derde partij, Eve, zich tussen Alice en Bob bevindt. Wanneer Alice informatie stuurt naar Bob, doet Eve zich voor als Bob. Omgekeerd, wanneer Bob informatie stuurt naar Alice, doet Eve zich voor als Alice. Op die manier kan Eve de private sleutel onderscheppen en alle geheime berichten, die op deze sleutel gebaseerd zijn, lezen.

We bekijken twee concrete voorbeelden van dit protocol. Het eerste voorbeeld 13 is het meest ideale geval: er worden geen qubits onderschept tussen Alice en Bob en het kwantumkanaal bevat geen ruis.

Voorbeeld 13 *Uitgewerkt voorbeeld van het BB84 protocol zonder afluisteren en/of ruis. In stap (5) vergelijken Alice en Bob de basisbits b en b'. Bits die niet overeenstemmen, worden niet opgenomen in het eindresultaat 'sleutel'.*

geval	*I*	*II*	*III*	*IV*	*V*	*VI*	*VII*	*VIII*
databits van Alice (a)	*1*	*1*	*0*	*1*	*0*	*1*	*1*	*1*
basisbits van Alice (b)	*0*	*1*	*0*	*1*	*1*	*0*	*1*	*0*
qubits van Alice (q)	$\|1\rangle$	$\|-\rangle$	$\|0\rangle$	$\|-\rangle$	$\|+\rangle$	$\|1\rangle$	$\|-\rangle$	$\|1\rangle$
meetbasis van Bob (b')	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*1*
databits van Bob (a')	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*1*
sleutel		*1*	*0*			*1*	*1*	

We zien dat Bob in de gevallen I, IV, V en VIII, de verkeerde berekeningsbasissen kiest. Dit betekent dat hij 1/2 kans heeft om de juiste databit a'_i te bekomen. We kunnen dit als volgt verklaren. Veronderstellen we dat de qubit $|\psi\rangle = |1\rangle$ geconstrueerd wordt op basis van de rechthoekige berekeningsbasissen. De probabiliteiten van de mogelijke meetresultaten ($m = 0$ voor $|0\rangle$ en $m = 1$ voor $|1\rangle$), bij meting in de rechthoekige berekeningsbasissen, kunnen we berekenen met behulp van formules (2.17) en (2.18) uit hoofdstuk 2.

$$p(0) = |0|^2 = 0, p(1) = |1|^2 = 1 \tag{7.37}$$

Wanneer we dezelfde qubit $|\psi\rangle$ meten in de (verkeerde) diagonale berekeningsbasissen ($m = 0$ voor $|+\rangle$ en $m = 1$ voor $|-\rangle$), krijgen we volgende probabiliteiten met behulp van formules (2.15) en (2.16).

$$p(0) = \frac{|0+1|^2}{2} = \frac{1}{2}, p(1) = \frac{|0-1|^2}{2} = \frac{1}{2} \tag{7.38}$$

Dit idee geldt ook in de omgekeerde richting: wanneer de qubit werd geconstrueerd in de diagonale berekeningsbasis, is er 1/2 kans dat Bob de

juiste databit a'_i bekomt. Met deze gegevens kunnen we de kans berekenen opdat een databit a_i foutloos wordt overgemaakt aan Bob. We onderscheiden twee gevallen waarbij de qubit wordt onderschept tijdens het versturen door een derde partij Eve.

- Eve meet de qubit toevallig in de juiste berekeningsbasis. Dit gebeurt in 1/2 van de gevallen.
- Eve meet de qubit in de verkeerde berekeningsbasis. Dit gebeurt eveneens in 1/2 van de gevallen. Nu heeft Bob nog 1/2 kans om alsnog de juiste databit te meten.

De totale probabiliteit om een enkele databit correct te ontvangen bedraagt $1/2 + 1/2 \cdot 1/2 = 3/4$. Met andere woorden, er is een continue foutprobabiliteit $\epsilon_n = 1/4^n$ voor een systeem met n qubits. Hoe groter het aantal qubits n, hoe lager de totale foutprobabiliteit gaat liggen.

7.4.2 Afluisteren van qubits

Voorbeeld 14 *Uitgewerkt voorbeeld van het BB84 protocol waarbij afluisteren en/of ruis in rekenschap werden gebracht. In gevallen II en III zijn de qubits vervalst en/of onderhevig aan ruis. Merk op dat qubits $|q_i\rangle = |q'_i\rangle$ mogelijks ook onderschept geweest zijn, maar de vervalsing is blijkbaar gelijk aan het origineel.*

geval	*I*	*II*	*III*	*IV*	*V*	*VI*	*VII*	*VIII*
databits van Alice (a)	*1*	*1*	*0*	*1*	*0*	*1*	*1*	*1*
basisbits van Alice (b)	*0*	*1*	*0*	*1*	*1*	*0*	*1*	*0*
qubits van Alice (q)	$\|1\rangle$	$\|-\rangle$	$\|0\rangle$	$\|-\rangle$	$\|+\rangle$	$\|1\rangle$	$\|-\rangle$	$\|1\rangle$
qubits van Bob (q′)	$\|1\rangle$	$\|0\rangle$	$\|+\rangle$	$\|-\rangle$	$\|0\rangle$	$\|1\rangle$	$\|-\rangle$	$\|1\rangle$
meetbasis van Bob (b′)	*1*	*1*	*0*	*1*	*0*	*1*	*1*	*1*
databits van Bob (a′)	*0*	*1*	*1*	*1*	*0*	*1*	*1*	*0*
sleutel		*1*	*1*	*1*			*1*	

We onderscheiden enkele merkwaardige gevallen.

- In geval II is de qubit $|q'_{II}\rangle$ veranderd van $|-\rangle$ naar $|0\rangle$. Bob heeft echter, met probabiliteit 1/2, de juiste databit $a'_{II} = 1$ gemeten. Omdat de meetbasis $b_{II} = b'_{II}$ correct is, zal dit resultaat niet geschrapt worden in stap (5). Geen enkel mechanisme zal deze fout kunnen detecteren.

- Geval III is volledig equivalent aan geval II, met uitzondering van het bekomen meetresultaat. Bob heeft hier de verkeerde databit $a'_{III} = 1$ gemeten. Het is mogelijk dat deze fout in de stappen (6) en (7) onderschept wordt. Enkel wanneer dergelijke fout té vaak zou optreden, moet het protocol herstart worden.

- In geval V is de qubit $|q'_V\rangle$ veranderd van $|+\rangle$ naar $|0\rangle$ én heeft Bob de verkeerde meetbasis $b_{II} \neq b'_{II}$ gekozen. Met probabiliteit 1/2 heeft hij alsnog de juiste databit $a_{II} = a'_{II}$ gemeten. Deze zal echter geschrapt worden wanneer Alice en Bob hun bits b en b' controleren in stap 5.

- Geval VII is ogenschijnlijk correct verlopen daar $|q_V\rangle = |q'_V\rangle$. Merk op dat de mogelijkheid bestaat dat de qubit correct werd afgeluisterd en dat door toeval een nieuwe juiste qubit wordt gegeven aan Bob. Dit verklaart het belang om na afloop van het BB84 protocol, een hashfunctie toe te passen op de private sleutels.

7.4.3 Implementatie

De implementatie van het BB84 protocol zou een gedistribueerd model vereisen [Sta05]. Wij beperken ons tot de programmacode om qubits te coderen en te decoderen volgens de rechthoekige of diagonale berekeningsbasissen.

De symbolen `value` en `alphabet` stellen respectievelijk a_i en b_i voor. Wanneer `alphabet` naar `nil` evalueert, wordt de qubit geconstrueerd in de rechthoekige berekeningsbasissen $|0\rangle$ en $|1\rangle$, in het andere geval worden de diagonale berekeningsbasissen gebruikt $|+\rangle$ en $|-\rangle$. Deze laatste worden geconstrueerd op basis van de Hadamard operator uit formules 2.12 en 2.13

```
(defun encode-qubit (value alphabet)
  "encodes a value (0 or 1) and an alphabet (nil or t)
   to a single qubit"
  (let ((_result_ (make-qureg (standard-init value))))
    (if alphabet
        (qc-apply-straight _result_ -h-)
      _result_)))
```

De functie `decode-qubit` meet de qubit in de berekeningsbasissen gespecificeerd door `alphabet`. Deze functie evalueert naar `t` als het meetresultaat $a'_i = 1$ en naar `nil` in het andere geval. Het meten in de diagonale berekeningsbasis wordt geïmplementeerd door eerst de Hadamard operator toe te passen.

```
(defun decode-qubit (_qureg_ alphabet)
  "decodes _qureg_ according to alphabet (nil or t)"
  (let ((_result_ (if alphabet
                      (qc-apply-straight -h- _qureg_)
                    _qureg_)))
    (= (collapse-basis _result_) 1)))
```

Hoofdstuk 8

Conclusies & verder onderzoek

8.1 Conclusies

De programmeertaal Lisp werd uitgebreid met de nodige abstracte datatypes om willekeurige kwantumberekeningen in uit te drukken. Op die manier ontstaat er een heterogeen karakter tussen klassiek en kwantum, vandaar dat we spreken over een hybride architectuur. We hebben gekozen om de kwantum programmeertaal volledig te integreren in Lisp, op die manier kunnen klassiek en kwantum elkaar wederzijds aanvullen.

In de kwantumsimulator kan men twee grote abstractielagen onderscheiden. De kwantum abstractielaag bevat de abstracte datatypes om kwantumberekeningen in uit te drukken. De klassieke abstractielaag is een soort van wiskundige bibliotheek die functies uit de lineaire algebra implementeert. De kwantumsimulator fungeert als bindmiddel tussen deze twee abstractielagen door kwantumberekeningen om te zetten in concepten uit de lineaire algebra. Dit verklaart waarom we de kwantumsimulator ook een modellering noemen.

Qubits worden geïmplementeerd door middel van kwantumregisters. Dit abstracte datatype worden intern voorgesteld als een geheugenvriendelijke matrixvoorstelling, gebaseerd op het idee van ijle matrices. De kwantumoperatoren zijn eveneens geïmplementeerd op basis van deze efficiënte matrixvoorstelling.

Kwantumregisters zijn voorzien van de nodige functies om de amplitudes van een qubit te kunnen raadplegen en te wijzigen. Deze experimenteermogelijkheid is niet realiseerbaar op werkelijke kwantumcomputers. Met het oog op gebruiksvriendelijkheid werden macro's en hulpfuncties ontwikkeld om kwantumregisters te initialiseren in willekeurige superposities. Een krachtig iteratiemechanisme gebaseerd op filters laat toe om efficiënt over bepaalde superposities van een qubit te itereren. Deze techniek is interessant om bij-

voorbeeld de simulatie van de partiële meting te implementeren.

De simulator beschikt over tal van optimalisaties om kwantumberekeningen te simuleren. Dergelijke optimalisaties zijn noodzakelijk als gevolg van de exponentiële complexiteit van de kwantummechanische natuur. We hebben aangetoond dat bepaalde kenmerken van kwantumoperatoren, zoals de Pauli-Z operator, een specifieke optimalisatie met zich meebrengen. De ideeën van specifieke optimalisaties worden gebundeld tot een algemene optimalisatie. Deze past een willekeurige kwantumoperator toe op een enkelvoudige qubit die deel uitmaakt van een meervoudige qubit.

Het innovatieve aan de kwantumsimulator is het gebruik van de multi-paradigma programmeertaal Lisp. We maakten volop gebruik van de programmeerkundige voordelen van Lisp om een hoog-niveau kwantum programmeertaal te ontwikkelen met een hoge uitdrukkingskracht. We zijn van mening dat expressiviteit code-duplicatie tegengaat, code compacter maakt en bijgevolg de leesbaarheid bevordert. We volgen daarbij de "van beneden naar boven" programmeerfilosofie van Lisp. Op die manier hebben we een taal gecreëerd die, zonder afgeremd te worden door een bepaald programmeerparadigma, nauw aansluit bij de problemen die we wensen uit te drukken. De all-round mogelijkheden van de kwantumsimulator werden geïllustreerd door diverse populaire kwantumtoepassingen te implementeren.

8.2 Verder onderzoek

De huidige implementatietoestand van de eigen kwantumsimulator staat te borrelen om nog verder uit te groeien. De programmacode is bijzonder compact, leesbaar en performant. Het zou een goede oefening zijn om nog een stap verder te gaan en de kwantumsimulator uit te bouwen tot een evaluator die het QRAM-model ondersteunt. We bemerken dat de meeste huidige kwantumsimulatoren de mogelijkheden van Lisp niet kunnen evenaren. Hun programmeerkundige mogelijkheden zijn beperkt door bijvoorbeeld statische typering of het gebruik van tweede-klasse objecten. We concluderen hieruit dat de ontwikkeling van hoog-niveau kwantum programmeertalen nog steeds in ontwikkeling is.

We werpen de schijnwerpers nog op een andere kwestie. Het valt op dat de huidige kwantum programmeertalen, de eigen kwantumsimulator incluis, opvallend veel gebruik maken van indices. Zo bestaan er situaties waar simulatiefuncties gemengd worden met ingewikkelde iteratiemechanismes om bepaalde kwantumcircuits te implementeren. Op die manier daalt de leesbaarheid van de programmacode zienderogen.

Een interessant onderwerp zou het onderzoek naar programmeerstijlen

kunnen zijn die nauwer aansluiten bij het uitdrukken van kwantumcircuits, en dus het gebruik van vervelende indices op een of andere manier weten te minimaliseren. Een goede oplossing zou kunnen zijn om een domein specifieke taal te ontwikkelen die de dynamiciteit van kwantumcircuits vastlegt. Met dynamiciteit bedoelen we bijvoorbeeld de patronen die steeds terug te vinden zijn in bepaalde kwantumcircuits, zoals de discrete Fourier transformatie.

Wanneer we de dynamiciteit of "patroon" van een kwantumcircuit kunnen afzonderen, komen we terecht op een hoger abstractieniveau. Een kwantumcircuit wordt dan beschreven door een patroon én de nodige kwantumoperatoren die ingevuld moeten worden in dat patroon. Deze manier van werken resulteert ongetwijfeld in minder code-duplicatie. Iemand kan immers een bibliotheek van vaak voorkomende patronen aanleggen om toekomstige kwantumcircuits mee te beschrijven. Deze manier van werken is tot op heden niet terug te vinden in bestaande kwantum programmeertalen.

Bijlage A

Wiskundige achtergrond

A.1 Lineaire algebra

A.1.1 Vectorruimtes en Dirac-notatie

De basisobjecten binnen de lineaire algebra zijn *vectorruimtes*. Voor onze doeleinden is enkel de complexe vectorruimte $\mathbb{C}^n$ van belang, dit is een verzameling van geordende n-tallen van de vorm $(a_1, a_2, \ldots, a_n)$ met $a_1, a_2, \ldots, a_n \in \mathbb{C}$. De elementen van een vectorruimte noemt men *vectoren*. Binnen de kwantummechanica wordt doorgaans de Dirac-notatie $|a\rangle$ gebruik om vectoren te noteren. Vectoren kunnen voorgesteld worden door middel van kolommatrices.

$$|a\rangle = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \tag{A.1}$$

Duaal van een vector

De *duaal* van een vector $|a\rangle$ kan berekend worden door de kolommatrix (A.1) te transponeren en de complex toegevoegde waarde $(*)$ van elk element te nemen. Het resultaat staat in formule (A.2). De duaal noteert men als $\langle a|$ in Dirac-notatie.

$$\langle a| = \begin{pmatrix} a_1* & a_2* & \cdots & a_n* \end{pmatrix} \tag{A.2}$$

Inwendig product van vectoren

Het *inwendig product* is een functie $(.,.)$ die twee vectoren $|a\rangle$ en $|b\rangle$ uit $\mathbb{C}^n$ als invoer neemt en een complex getal $(|a\rangle, |b\rangle)$ berekent. De standaard kwantummechanische Dirac-notatie voor dit product is $\langle a|b\rangle$. Het inwendig product moet voldoen aan volgende voorwaardes.

- $(|a\rangle, \sum_i \lambda_i |w_i\rangle) = \sum_i \lambda_i (|a\rangle, |b_i\rangle)$
- $\langle a|b\rangle = \langle b|a\rangle *$
- $\langle a|a\rangle \geq 0 \Leftrightarrow |a\rangle = 0$

Het inwendig product van de complexe vectorruimte $\mathbb{C}^n$ is als volgt gedefinieerd.

$$\langle a|b\rangle = (a,b) = \sum_i a_i^* b_i = \begin{pmatrix} a_1^* & \cdots & a_n^* \end{pmatrix} \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} \tag{A.3}$$

Kenmerken van vectoren

Op basis van het inwendig product, kan men volgende kenmerken definiëren.

- Twee vectoren $|a\rangle$ en $|b\rangle$ zijn *orthogonaal* desda $\langle a|b\rangle = 0$.
- De *norm* van een vector is $\||a\rangle\| = \sqrt{\langle a|a\rangle}$.
- Een vector met norm 1, noemt met een *eenheidsvector*.
- Een verzameling van vectoren $A = \{|0\rangle, |1\rangle, \ldots, |n-1\rangle\}$ is *orthonormaal* als voor alle mogelijke paren $|i\rangle$ en $|j\rangle$ uit de verzameling A geldt dat $\langle i|j\rangle = \delta_{ij}$. Het symbool δ_{ij} heet men de *Kronecker delta*, deze is als volgt gedefinieerd.

  $$\delta_{ij} = \begin{cases} 1 & \text{als } i = j \\ 0 & \text{als } i \neq j \end{cases}$$

  De voorwaarde dat $\delta_{ij} = 1$ wanneer $i = j$ geeft aan dat alle vectoren uit A eenheidsvectoren moeten zijn. De andere voorwaarde, dat $\delta_{ij} = 0$ wanneer $i \neq j$ geeft aan dat alle vectoren orthogonaal moeten zijn.

Hilbert-ruimte $\mathcal{H}$

Alle kwantumberekeningen gebeuren in de Hilbert-ruimte $\mathcal{H}$, dit is een eindige complexe vectorruimte $\mathbb{C}^n$ die voorzien is van een inwendig product zoals hierboven staat beschreven.

A.1.2 Optelling en scalaire vermenigvuldiging

De opteloperator is gedefinieerd voor vectoren uit $\mathbb{C}^n$.

$$|a\rangle + |b\rangle = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \vdots \\ a_n + b_n \end{pmatrix} \tag{A.4}$$

De scalaire vermenigvuldiging voor $z \in \mathbb{C}$ gebeurt als volgt.

$$z|a\rangle = \begin{pmatrix} za_1 \\ za_2 \\ \vdots \\ za_n \end{pmatrix} \tag{A.5}$$

De optelling en scalaire vermenigvuldiging binnen de complexe vectorruimte $\mathbb{C}^n$ voldoen aan volgende eigenschappen.

- Met betrekking tot de optelling.
  - $|a\rangle + |b\rangle \in \mathbb{C}^n$
  - $(|a\rangle + |b\rangle) + |c\rangle = |a\rangle + (|b\rangle) + |c\rangle) = |a\rangle + |b\rangle + |c\rangle$
  - $\exists |0\rangle \in \mathbb{C}^n : |0\rangle + |a\rangle = |a\rangle + |0\rangle = |a\rangle$
  - $\forall |a\rangle \in \mathbb{C}^n, \exists - |a\rangle \in \mathbb{C}^n : |a\rangle + -|a\rangle = |0\rangle$
  - $|a\rangle + |b\rangle = |b\rangle + |a\rangle$
- Met betrekking tot de scalaire vermenigvuldiging.
  - $z|a\rangle \in \mathbb{C}^n$
  - $(z + m)|a\rangle = z|a\rangle + m|a\rangle$
  - $z(m|a\rangle) = (zm)|a\rangle$
  - $z(|a\rangle + |b\rangle) = z|a\rangle + z|b\rangle$
  - $1 \cdot |a\rangle = |a\rangle$

A.1.3 Lineaire combinaties

Een *lineaire combinatie* van de vectoren $|a_1\rangle, |a_2\rangle, \ldots, |a_p\rangle$ is elke vector $|v\rangle$ zoals hieronder weergegeven waarvoor geldt dat $r_i \in \mathbb{C}$ voor $1 \leq i \leq p$.

$$|v\rangle = r_1|a_1\rangle + r_2|a_2\rangle + r_3|a_3\rangle + \ldots + r_p|a_p\rangle \tag{A.6}$$

Voortbrengende verzameling

Men noemt $B = \{|a_1\rangle, |a_2\rangle, \ldots, |a_p\rangle\}$ een *voortbrengende verzameling* desda elke vector van $\mathbb{C}^n$ te schrijven is als een lineaire combinatie van de vectoren $|a_1\rangle, |a_2\rangle, \ldots, |a_p\rangle$. Bijvoorbeeld, het is mogelijk om elke vector $|\psi\rangle$ in $\mathbb{C}^2$ te schrijven als een lineaire combinatie $|\psi\rangle = a|0\rangle + b|1\rangle$ van de vectoren $|0\rangle = (1, 0)$ en $|1\rangle = (0, 1)$. Daarbij geldt dat $a, b \in \mathbb{C}$.

Lineair onafhankelijke verzamelingen

De verzameling $B = \{|a_1\rangle, |a_2\rangle, \ldots, |a_p\rangle\}$ is lineair onafhankelijk desda $r_1|a_1\rangle + r_2|a_2\rangle + r_3|a_3\rangle + \ldots + r_p|a_p\rangle = |0\rangle \Rightarrow r_1 = r_2 = \ldots = r_p = 0$.

Basisvectoren

De verzameling $B = \{|a_1\rangle, |a_2\rangle, \ldots, |a_p\rangle\}$ is een basis van de complexe vectorruimte $\mathbb{C}^n$ desda B een voortbrengende verzameling is voor $\mathbb{C}^n$ en B lineair onafhankelijk is. Bijvoorbeeld, de basistoestanden van willekeurige qubits zijn steeds orthonormale basisvectoren.

A.1.4 Lineaire operatoren

We noemen A van $\mathbb{C}^n$ naar $\mathbb{C}^n$ een lineaire operator desda het beeld van de lineaire combinatie van vectoren gelijk is aan de lineaire combinatie van de beelden. Formeel betekent dit voor $|a\rangle, |b\rangle \in \mathbb{C}^n$ en $u, v \in \mathbb{C}$:

$$A : \mathbb{C}^n \to \mathbb{C}^n \text{ is een lineaire operator } \Leftrightarrow A(u|a\rangle + v|b\rangle) = uA(|a\rangle) + vA(|b\rangle) \tag{A.7}$$

Eigenschappen van lineaire operatoren

- $A|0\rangle = |0\rangle$
- $A(|a\rangle + |b\rangle) = A|a\rangle + A|b\rangle$

- $A(z|a\rangle) = zA|a\rangle$

Een lineaire afbeelding A is volledig bepaald als men de beelden van de basisvectoren kent. Elke vector is immers te schrijven in functie van die basisvectoren.

Matrix van een lineaire operator

Beschouwen we de lineaire operator $A : \mathbb{C}^n \to \mathbb{C}^n$. De beelden van de basisvectoren $|0\rangle, |1\rangle, \ldots, |n-1\rangle$ kan men schrijven in functie van dezelfde basisvectoren.

$$\begin{aligned} A|0\rangle &= a_{00}|0\rangle + a_{10}|1\rangle + \ldots + a_{(n-1)1}|n-1\rangle \\ A|1\rangle &= a_{01}|0\rangle + a_{11}|1\rangle + \ldots + a_{(n-1)2}|n-1\rangle \\ &\vdots \\ A|n-1\rangle &= a_{0(n-1)}|0\rangle + a_{1(n-1)}|1\rangle + \ldots + a_{(n-1)(n-1)}|n-1\rangle \end{aligned} \quad \text{(A.8)}$$

De matrix van de lineaire operator tegenover de basisvectoren $|0\rangle$, $|1\rangle$, …, $|n-1\rangle$ ziet er als volgt uit. Elke kolom bevat de coëfficiënten van het beeld van een basisvector van $\mathbb{C}^n$.

$$\begin{pmatrix} a_{00} & a_{10} & \cdots & a_{0(n-1)} \\ a_{10} & a_{11} & \cdots & a_{1(n-1)} \\ \vdots & \vdots & & \vdots \\ a_{(n-1)0} & a_{(n-1)1} & \cdots & a_{(n-1)(n-1)} \end{pmatrix} \quad \text{(A.9)}$$

Hermitiaanse operatoren

Beschouwen we een willekeurige lineaire operator A in de complexe vectorruimte $\mathbb{C}^n$. Voor elke lineaire operator A bestaat er een unieke *hermitiaanse operator* $A^\dagger$. Dergelijke operator kan berekend worden door de *adjunct* ($\dagger$) van de matrixvoorstelling te nemen, dit is de samenstelling $(* \circ T)$ van een transponatie (T) en het nemen van de complex toegevoegde waarde $(*)$ van elk element uit de matrix A.

$$A^\dagger = (A^T)^* \quad \text{(A.10)}$$

Beschouwen we een lineaire operator A die gedefinieerd is over $\mathbb{C}^n \to \mathbb{C}^n$ en twee vectoren $|a\rangle, |b\rangle \in \mathbb{C}^n$. Daarvoor geldt dat:

$$\langle a|Ab\rangle = \langle A^\dagger a|b\rangle \quad \text{(A.11)}$$

Unitaire operatoren

Een unitaire transformatie binnen $\mathbb{C}^n$ kan voorgesteld worden door een vierkante matrix met dimensie $2^n \times 2^n$ die aan volgende voorwaarde voldoet.

$$U^\dagger U = UU^\dagger = I \tag{A.12}$$

Het belangrijkste kenmerk van unitaire operatoren is dat ze orthonormaliteit van de basisvectoren bewaren.

A.1.5 Tensorproduct van vectorruimtes

Het *tensorproduct* is een manier om twee vectorruimtes samen te voegen. Veronderstellen we de complexe vectorruimtes $\mathbb{C}^n$ en $\mathbb{C}^m$. Het tensorproduct $\mathbb{C}^n \otimes \mathbb{C}^m$ resulteert in de vectorruimte $\mathbb{C}^{nm}$. Volgende eigenschappen zijn van toepassing.

- Voor $z \in \mathbb{C}$, $|a\rangle \in \mathbb{C}^n$ en $|b\rangle \in \mathbb{C}^m$ geldt dat $z(|a\rangle \otimes |b\rangle) = (z|a\rangle) \otimes |b\rangle = |a\rangle \otimes (z|b\rangle)$.
- Voor $|a_1\rangle, |a_2\rangle \in \mathbb{C}^n$ en $|b\rangle \in \mathbb{C}^m$ geldt dat $(|a_1\rangle + |a_2\rangle) \otimes |b\rangle = |a_1\rangle \otimes |b\rangle + |a_2\rangle \otimes |b\rangle$.
- Voor $|a\rangle \in \mathbb{C}^n$ en $|b_1\rangle, |b_2\rangle \in \mathbb{C}^m$ geldt dat $|a\rangle \otimes (|b_1\rangle + |b_2\rangle) = |a\rangle \otimes |b_1\rangle + |a\rangle \otimes |b_2\rangle$.

Het tensorproduct van de orthogonale basisvectoren van twee complexe vectorruimtes $\mathbb{C}^n$ en $\mathbb{C}^m$, zullen fungeren als orthogonale basisvectoren voor $\mathbb{C}^{mn}$. De Dirac-notatie voor het tensorproduct van twee vectoren $|a\rangle \otimes |b\rangle$ kan vereenvoudigd worden tot $|a, b\rangle$ of $|ab\rangle$.

We verklaren hoe het tensorproduct concreet kan berekend worden door de matrixvoorstelling te gebruiken. Beschouwen we een $m \times n$ matrix A en een $p \times q$ matrix B. Het (Kronecker) tensorproduct $A \otimes B$ is een $mp \times nq$ matrix die staat weergegeven in formule (A.13). Elk element aB uit $A \otimes B$ is een submatrix C met elementen $c(i, j) = a \cdot b(i, j)$.

$$A \otimes B = \begin{pmatrix} a(1,1)B & a(1,2)B & \cdots & a(1,n)B \\ a(2,1)B & a(2,2)B & \cdots & a(2,n)B \\ \vdots & \vdots & \vdots & \vdots \\ a(m,1)B & a(m,2)B & \cdots & a(m,n)B \end{pmatrix} \tag{A.13}$$

A.2 Spooroperator

A.2.1 Definitie

Het spoor, aangegeven met tr, van een vierkante matrix is de som van de diagonaalelementen.

$$tr(A) = \sum_i A_{ii} \tag{A.14}$$

A.2.2 Eigenschappen

Deze spooroperator voldoet aan volgende eigenschappen met $z \in \mathbb{C}$.

- $tr(AB) = tr(BA)$
- $tr(A + B) = tr(A) + tr(B)$
- $tr(zA) = z \cdot tr(A)$

A.2.3 Gevolgen

Beschouwen we een eenheidsvector $|\psi\rangle \in \mathbb{C}^n$ en een willekeurige operator A van $\mathbb{C}^n$ naar $\mathbb{C}^n$. Men kan bewijzen dat er voor $|\psi\rangle$ een verzameling O met orthonormale basisvectoren $|i\rangle$ bestaat. De verzameling O bevat onder andere de vector $|\psi\rangle$. Er geldt:

$$\begin{aligned} tr(A|\psi\rangle\langle\psi|) &= \sum_i \langle i|A|\psi\rangle\langle\psi|i\rangle \\ &= \langle\psi|A|\psi\rangle \end{aligned} \tag{A.15}$$

A.3 Kettingbreuken

We ontwikkelen een techniek om met breuken een willekeurig rationaal getal te benaderen.

A.3.1 Definitie

Elk getal $\nu \in \mathbb{Q}$ kan geschreven worden als onderstaande eindigde expressie met $a_0 \in \mathbb{N}$ en $a_1, \ldots, a_n \in \mathbb{N}_0$.

$$\nu = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ldots + \cfrac{1}{a_n}}}}} \tag{A.16}$$

Formule (A.16) noemt men een *kettingbreuk*, deze worden vaak als volgt genoteerd $[a_0, a_1, \ldots, a_n]$. De elementen van een kettingbreuk kunnen berekend worden met behulp van onderstaande recurrentie relaties.

$$a_0 = \lfloor \nu \rfloor \tag{A.17}$$
$$\nu_0 = \nu - a_0 \tag{A.18}$$
$$a_{n+1} = \lfloor 1/\nu_n \rfloor \tag{A.19}$$
$$\nu_{n+1} = \frac{1}{\nu_n} - a_{n+1} \tag{A.20}$$

A.3.2 Benaderende breuken

De i-de benaderende breuk p_i/q_i is de kettingbreuk $\nu_i = [a_0, a_1, \ldots, a_i]$. Merk op dat de n-de benaderende breuk overeenstemt met ν. De gehele getallen p_i en q_i kunnen met de volgende recurrentie relaties berekend worden.

$$p_0 = a_0,\ p_1 = a_1 a_0 + 1,\ p_n = a_n p_{n-1} + p_{n-2} \tag{A.21}$$

$$q_0 = 1,\ q_1 = a_1,\ q_n = a_n q_{n-1} + q_{n-2} \tag{A.22}$$

Bijlage B

RSA Encryptie

B.1 Concept

Een publiek encryptiesysteem onderscheidt zich van een privaat cryptosysteem daar er twee encryptiesleutels (publieke en private) aan te pas komen. Het RSA [RSA78] encryptieproces verloopt informeel als volgt tussen zender A en ontvanger B met bericht M.

1. Partij A versleutelt zijn bericht M met de publieke sleutel van de tegenpartij B.
2. Het versleutelde bericht M wordt verstuurd naar partij B.
3. Partij B heft de versleuteling op door zijn private sleutel toe te passen op het versleutelde bericht M.

De theorie van publieke cryptosystemen, uitgevonden door Diffie en Hellman in 1976, geeft aan dat de publieke en private sleutel elkaars inverse moeten zijn. Daarbij mag het niet 'praktisch' haalbaar zijn om de private sleutel af te leiden uit de publieke sleutel. RSA heeft deze theorie geconcretiseerd vanuit de idee dat priemgetallen efficiënt kunnen berekend worden, maar niet gefactoriseerd. Dit laatste is echter een aanname. Tot op heden is het niet mogelijk om efficiënt een priemfactorisatie te realiseren met behulp van klassieke berekeningen.

B.2 Uitgewerkt voorbeeld

Op basis van een vereenvoudigd en onrealistisch voorbeeld, illustreren we de praktische werkwijze van het RSA cryptosysteem. Voor een gedetailleerde

wiskundige achtergrond, verwijzen we naar [RSA78]. We zoomen in op de generatie van de private sleutel en preciseren de rol van een priemfactorisatie om RSA te kraken.

We genereren eerst een publieke sleutel op basis van de willekeurig gegenereerde priemgetallen $p = 53$ en $q = 59$. Dit kan op een niet-probabilistische manier gebeuren in polynomiale rekentijd met behulp van het algoritme van blabla [x].

$$n = pq = 53 \cdot 59 = 3127 \tag{B.1}$$

De encryptie-exponent e kan met bovenstaande gegevens als volgt berekend worden. Dit kan zonder een noewenswaardig performantieverlies berekend worden via trial-and-error, daar de dichtheid van relatieve priemen heel hoog ligt.

$$\text{ggd}(e, (p-1)(q-1)) = 1 \Leftrightarrow e = 11 \tag{B.2}$$

De publieke sleutel ziet er nu als volgt uit.

$$(e, n) = (11, 3127) \tag{B.3}$$

De bedoeling is nu om het bericht $M =$"OK"$= 1511$ te versleutelen. Dit gebeurt op basis van de publieke sleutel $(e, n) = (11, 3127)$ uit formule (B.3). Het versleuteld bericht C kan als volgt berekend worden.

$$C = M^e \bmod n = 1511^{11} \bmod 3127 = 2742 \tag{B.4}$$

Om het versleutelde bericht $C = 2742$ uit formule (B.4) te kunnen ontsleutelen, hebben we een private sleutel nodig. Deze laatste kan berekend worden op basis van de priemfactoren $p = 53$, $q = 59$ en de encryptie-exponent $e = 11$. De berekening van de decryptie-exponent d gebeurt met onderstaande formule.

$$ed = 1 \bmod (p-1)(q-1) \Leftrightarrow d = 1317 \tag{B.5}$$

De private sleutel ziet er nu als volgt uit.

$$(d, n) = (1317, 3127) \tag{B.6}$$

Het kraken van RSA gebeurt in dit stadium. Veronderstel dat we enkel over de publieke sleutel $(e, n) = (11, 3127)$ beschikken. We kunnen hieruit de variabelen p en q afleiden omdat we weten dat het priemgetallen zijn. De hoofdstelling van de rekenkunde zegt dat elk getal $n \in \mathbb{N} : n > 1$ op juist één manier te ontbinden is in onderling verschillende priemfactoren en hun

machten. Een priemfactorisatie van n zal dus met zekerheid de variabelen p en q opleveren, wat ons in staat stelt om formule (B.5) te berekenen.

In de laatste stap ontsleutelen we het versleutelde bericht $C = 2742$ uit formule (B.4). We gebruiken daarvoor de private sleutel $(d, n) = (1317, 3127)$ uit formule (B.6). De berekening ziet er als volgt uit.

$$M = C^d \bmod n = 2742^{1317} \bmod 3127 = 1511 = \text{“OK”} \tag{B.7}$$

Bibliografie

[AS87] Harold Abelson and Gerald Jay Sussman. Lisp: A language for stratified design. *AI Lab Memo AIM-986*, 1987.

[AwJS96] Harold Abelson and Gerald Jay Sussman with Julie Sussman. *Structure and interpretation of computer programs.* The MIT Press, 1996.

[BB84] C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE international Conference on Computers, Systems and Signal Processing, Bangalore, India*, page 175, New York, 1984. IEEE Press.

[Ben73] C. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 17:5225, 1973.

[Ben82] P. Benioff. Quantum mechanical Hamiltonian models of Turing machines. *J. of Stat. Phys.*, 29:515, 1982.

[Ber04] Frederik Vanden Berghe. The development of a high-level quantum programming language by extending pico. Master's thesis, Vrije Universiteit Brussel, 2004.

[Bet03] Stefano Bettelli. Toward an architecture for quantum programming. *Eur. Phys. J. D*, 25(2):181–200, 2003.

[BV97] Ethan Bernstein and Umesh V. Vazirani. Quantum complexity theory. volume 26, pages 1411–1473, 1997.

[CEMM98] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. Quantum algorithms revisited. *Proc. R. Soc. of Lond. A*, 454:339–354, 1998.

[Chu36] Alonzo Church. A note on the entscheidungsproblem. *Am. J. Math.*, 58:345–363, 1936.

[CT65] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:251–280, 1965.

[Deu85] David Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proc. R. Soc. Lond.*, pages A400: 97–117, 1985.

[Dir47] P. A. M. Dirac. *The Principles of Quantum Mechanics*. The international series of monographs on physics. Clarendon Press, Oxford, 4 edition, 1947.

[DJ92] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proc. R. Soc. Lond. A*, pages 439–553, 1992.

[DM] Theo D'Hondt and Wolfgang De Meuter. The pico project.

[Fey82] R. P. Feynman. Simulating physics with computers. *Int. J. of Theor. Phys.*, 21:467, 1982.

[GN96] R. B. Griffiths and C.-S. Niu. Semiclassical fourier transform for quantum computation. *Phys. Rev. Lett.*, 76:3228, 1996.

[Gra95] Paul Graham. *ANSI Common Lisp*. Prentice-Hall, 1995.

[Kit97] A. Y. Kitaev. Quantum error correction with imperfect gates. In O. Hirota, A. S. Holevo, and C. M. Caves, editors, *Proceedings of the third International Conference on Quantum Communication and Measurement*, New York, 1997. Plenum.

[Kni96] E. Knill. Conventions for quantum pseudocode, 1996. programming.

[Lan27] L. Landau. Das dämpfungsproblem in der wellenmechanik. *Z. Phys.*, (45):430–441, 1927.

[Lan01] Eric Landquist. The quadratic sieve factoring algoritm. 2001.

[Lim] The Harlequin Group Limited. Common lisp hyperspec (tm).

[Mag] Magiqtech. Magiq qpn(tm) security gateway, `http://www.magiqtech.com`.

[McC78] John McCarthy. History of lisp. In *HOPL-1: The first ACM SIGPLAN conference on History of programming languages*, pages 217–223, New York, NY, USA, 1978. ACM Press.

[MN] MathWorks and NIST. Jama: A java matrix package.

[Ö98] Bernhard Ömer. *A Procedural Formalism for Quantum Computing*. PhD thesis, Technical University of Vienna, 1998.

[Ö00] Bernhard Ömer. *Quantum Programming in QCL*. PhD thesis, Technical University of Vienna, 2000.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method of obtaining digital signatures and public-key cryptosystems. 21:120–126, 1978.

[SG96] Guy L. Steele and Richard P. Gabriel. The evolution of lisp. pages 233–330, 1996.

[Sho96] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comp.*, 26(5):1484–1509, 1996.

[Sta05] Dieter Standaert. Unpublished work on distributed quantum simulation. Master's thesis, Vrije Universiteit Brussel, 2005.

[Ste84] Guy Steele. *Common Lisp, the language*. Digital Press, 1984.

[Ste89] Guy Steele. *Common Lisp, the language, 2nd edition*. Digital Press, 1989.

[Ton03a] André Van Tonder. A lambda calculus for quantum computation. 2003.

[Ton03b] André Van Tonder. Quantum computation, categorical semantics and linear logic. 2003.

[Tur36] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. Lond. Math. Soc. 2*, 42:230–265, 1936.

[Uni] The Metaphysics Research Lab Stanford University. Stanford encyclopedia of philosophy, `http://plato.stanford.edu/`.

[vN27] J. von Neumann. Thermodynamik quantummechanischer Gesamtheiten. *Gött. Nach.*, 1:245, 1927.

[vN32] J. von Neumann. *Mathematische Grundlagen der Quantenmechanik*. Julius Springer-Verlag, 1932.

[Wal99] Julia Wallace. Quantum computer simulators - a review (version 2.1). 1999.

[Yao93] A. Yao. Quantum circuit complexity. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 352–360, Los Alamitos, California, 1993. IEEE Press.