

The QRL Book

**From Quantum Foundations to Causal Structure as a Programming
Primitive**

David Coldeira

2026-10-03

Table of contents

Welcome	1
The idea in one paragraph	1
What you will learn	2
Prerequisites	2
How to use this book	3
The code	3
About the author	4
License	4
 I. Part I: Quantum Foundations	 5
 1. Quantum Mechanics Basics	 7
1.1. The Quantum State	7
1.1.1. Mathematical Representation	7
1.1.2. General Qubit State	8
1.2. The Superposition Principle	8
1.2.1. Example: The Hadamard State	8
1.3. Measurement and the Born Rule	9
1.3.1. The Born Rule	9
1.3.2. Example: Measuring $ +\rangle$	9
1.4. Normalization Condition	10
1.4.1. Inner Product and Normalization	11
1.5. Measurement in Different Bases	11
1.5.1. Z-Basis vs X-Basis	11
1.5.2. Example: Measuring $ 0\rangle$ in X-Basis	12

Table of contents

1.6.	Global Phase and Physical Equivalence	12
1.6.1.	Example: Relative Phase Matters	13
1.7.	Quantum State Space: The Bloch Sphere	14
1.8.	Pure States vs Mixed States	15
1.9.	Summary	15
1.10.	Exercises	15
1.10.1.	Exercise 1.1: Normalization	15
1.10.2.	Exercise 1.2: Measurement Probabilities	16
1.10.3.	Exercise 1.3: Basis Transformation	16
1.10.4.	Exercise 1.4: Cross-Basis Measurement	17
1.10.5.	Exercise 1.5: Global Phase	17
1.10.6.	Exercise 1.6: Bloch Sphere Coordinates	17
2.	Entanglement	19
2.1.	Two-Qubit Systems	19
2.1.1.	Tensor Product	19
2.1.2.	Computational Basis for Two Qubits	20
2.1.3.	General Two-Qubit State	20
2.2.	Separable vs Entangled States	20
2.2.1.	Separable States	21
2.2.2.	Entangled States	21
2.3.	Bell States: Maximally Entangled Pairs	22
2.3.1.	Why Can't Bell States Be Factored?	22
2.4.	Physical Meaning of Entanglement	23
2.5.	Schmidt Decomposition	24
2.5.1.	Schmidt Rank and Entanglement	24
2.5.2.	Example: Schmidt Decomposition of $ \Phi^+\rangle$	24
2.5.3.	Example: Separable State	25
2.6.	Entanglement Entropy	25
2.6.1.	Reduced Density Matrix	25
2.6.2.	Von Neumann Entropy	26
2.6.3.	Example: Entropy of $ \Phi^+\rangle$	26
2.6.4.	Example: Partially Entangled State	27

Table of contents

2.7.	Bell's Theorem and Non-Locality	28
2.7.1.	EPR Paradox	28
2.7.2.	Bell's Inequality	28
2.8.	Measurement of Entangled States	29
2.8.1.	Example: Measuring $ \Phi^+\rangle$	29
2.8.2.	Partial Measurement Formula	30
2.9.	Summary	30
2.10.	Exercises	31
2.10.1.	Exercise 2.1: Identifying Entanglement	31
2.10.2.	Exercise 2.2: Schmidt Decomposition	31
2.10.3.	Exercise 2.3: Entanglement Entropy	31
2.10.4.	Exercise 2.4: Bell State Transformations	32
2.10.5.	Exercise 2.5: Partial Measurement	32
2.10.6.	Exercise 2.6: Creating Bell States	32
3.	n-Qubit Systems	35
3.1.	Tensor Products for n Qubits	35
3.1.1.	The n-Qubit Hilbert Space	35
3.1.2.	General n-Qubit State	35
3.2.	GHZ States: Maximal Multipartite Entanglement	36
3.2.1.	Definition	36
3.2.2.	Properties of GHZ States	36
3.2.3.	GHZ Correlations	37
3.2.4.	Entanglement Entropy of GHZ	37
3.3.	W States: Robust Entanglement	38
3.3.1.	Definition	38
3.3.2.	Properties of W States	38
3.3.3.	W State Resilience	39
3.3.4.	Entanglement Entropy of W	39
3.4.	GHZ vs W: Different Entanglement Classes	40
3.4.1.	LOCC Equivalence	40
3.4.2.	Summary of Differences	40
3.5.	Multipartite Entanglement: Beyond 2 Parties	41
3.5.1.	Genuine Multipartite Entanglement	41

Table of contents

3.5.2. Biseparable States	41
3.6. Measuring n-Qubit States	42
3.6.1. Partial Measurements	42
3.6.2. Example: Measuring GHZ_3	42
3.6.3. Example: Measuring W_3	43
3.7. Tensor Network Visualization	43
3.7.1. Product State	43
3.7.2. GHZ State	44
3.7.3. W State	44
3.8. Computational Complexity	44
3.8.1. State Vector Simulation	44
3.8.2. Tensor Network Methods	45
3.9. Creating n-Qubit States in QRL	45
3.10. Summary	46
3.11. Exercises	46
3.11.1. Exercise 3.1: GHZ State Verification	46
3.11.2. Exercise 3.2: W State Properties	47
3.11.3. Exercise 3.3: Entanglement Entropy Calculation	47
3.11.4. Exercise 3.4: LOCC Impossibility	47
3.11.5. Exercise 3.5: Multipartite Correlation	48
3.11.6. Exercise 3.6: Fragility vs Robustness	48
3.11.7. Exercise 3.7: n-Qubit Generalization	48
II. Part II: Measurement-Based Quantum Computing	49
4. MBQC Foundations	51
4.1. The Circuit Model vs MBQC	51
4.1.1. Traditional Circuit Model	51
4.1.2. Measurement-Based Quantum Computing	52
4.2. The One-Way Quantum Computer	52
4.3. Cluster States: The Universal Resource	53
4.3.1. Definition	53
4.3.2. Example: 3-Qubit Chain	53

Table of contents

4.3.3. Cluster State Properties	54
4.4. Measurement Patterns	54
4.4.1. Pattern Specification	54
4.4.2. Measurement Bases	55
4.4.3. Adaptive Measurements	55
4.5. Example: Single-Qubit Rotation	56
4.5.1. Setup	56
4.5.2. Protocol	56
4.6. Universality of MBQC	57
4.6.1. Why This Matters	57
4.7. Gate Teleportation	57
4.7.1. Bell Measurement Teleportation	58
4.7.2. Teleportation with Rotated Basis	58
4.8. Measurement Calculus	58
4.8.1. Commands	59
4.8.2. Example Pattern	59
4.9. Advantages of MBQC	60
4.9.1. For Photonic Quantum Computing	60
4.9.2. For Quantum Programming	60
4.10. MBQC vs Circuits: Complexity Comparison	60
4.11. Why MBQC Matters for Theory	61
4.12. Summary	62
4.13. Exercises	62
4.13.1. Exercise 4.1: Cluster State Preparation	62
4.13.2. Exercise 4.2: Measurement Outcomes	62
4.13.3. Exercise 4.3: Gate Teleportation	63
4.13.4. Exercise 4.4: Pattern Design	63
4.13.5. Exercise 4.5: Resource Scaling	63
4.13.6. Exercise 4.6: Advantage Analysis	63
5. Cluster States	65
5.1. Graph States: General Framework	65
5.1.1. Definition	65
5.1.2. Examples	66

Table of contents

5.2.	Cluster States vs General Graph States	66
5.3.	The Stabilizer Formalism	67
5.3.1.	Pauli Operators	67
5.3.2.	n-Qubit Pauli Group	67
5.3.3.	Stabilizers	68
5.3.4.	Stabilizer States	68
5.4.	Graph State Stabilizers	69
5.4.1.	Example: 2-Qubit Chain	69
5.4.2.	Example: 3-Qubit Chain	70
5.5.	Why Stabilizers Are Powerful	70
5.5.1.	Efficient Classical Representation	70
5.5.2.	Example: 100-Qubit Cluster	71
5.5.3.	Efficient Simulation	71
5.6.	Local Complementation	72
5.6.1.	Example	72
5.7.	Creating Cluster States in Practice	73
5.7.1.	Deterministic Preparation	73
5.7.2.	Photonic Preparation: Fusion Gates	74
5.8.	Measuring Graph States	74
5.8.1.	Single-Qubit Measurement	74
5.8.2.	Example: Measuring 3-Qubit Chain	75
5.8.3.	Adaptive Measurements	75
5.9.	Stabilizers Under Measurement	76
5.10.	2D Cluster States and Universality	76
5.11.	Summary	77
5.12.	Exercises	77
5.12.1.	Exercise 5.1: Stabilizer Verification	77
5.12.2.	Exercise 5.2: 4-Qubit Chain Stabilizers	77
5.12.3.	Exercise 5.3: Stabilizer Commutation	78
5.12.4.	Exercise 5.4: Local Complementation	78
5.12.5.	Exercise 5.5: Measurement Update	78
5.12.6.	Exercise 5.6: Graph State Entanglement	78
5.12.7.	Exercise 5.7: Resource Comparison	79

6. Measurement Patterns	81
6.1. From Circuits to Patterns	81
6.1.1. The Compilation Problem	81
6.2. Gate Teleportation: The Core Technique	82
6.2.1. Standard Teleportation Recap	82
6.2.2. Teleporting Through a Gate	82
6.3. Compilation Rules for Single-Qubit Gates	83
6.3.1. Rotations via Measurement Angles	83
6.3.2. Pauli Gates	83
6.3.3. Hadamard Gate	84
6.3.4. Phase Gate ($S = R_Z(\pi/2)$)	84
6.3.5. General Single-Qubit Unitary	85
6.4. Compilation Rules for Two-Qubit Gates	85
6.4.1. CNOT Gate	85
6.4.2. CZ Gate (Controlled-Z)	86
6.5. Full Circuit Compilation	87
6.5.1. Step-by-Step Process	87
6.5.2. Example: Simple Circuit	88
6.6. Adaptive Measurement	89
6.6.1. Feed-Forward Rules	89
6.6.2. Why This Works	89
6.7. Pauli Corrections	90
6.7.1. Correction Functions	90
6.7.2. Classical Post-Processing	90
6.8. Optimization Techniques	91
6.8.1. Minimizing Cluster Size	91
6.8.2. Minimizing Measurement Depth	91
6.8.3. Deterministic Patterns	92
6.9. From Patterns to QRL	92
6.9.1. QRL Abstraction Levels	92
6.10. Measurement Pattern Language	93
6.10.1. Commands	93
6.10.2. Example Pattern	94
6.11. Equivalence to Circuit Model	94

Table of contents

6.12. Summary	95
6.13. Exercises	95
6.13.1. Exercise 6.1: Hadamard Compilation	95
6.13.2. Exercise 6.2: Rotation Angle	95
6.13.3. Exercise 6.3: CNOT Pattern	96
6.13.4. Exercise 6.4: Adaptive Measurement	96
6.13.5. Exercise 6.5: Pattern Optimization	96
6.13.6. Exercise 6.6: Deterministic Gates	97
6.13.7. Exercise 6.7: Circuit to Pattern	97
 III. Part III: QRL Programming	 99
 7. Your First Bell State	 101
7.1. What is a Bell State?	101
7.2. Setting Up	101
7.3. Creating Quantum Systems	102
7.4. Entangling Systems	102
7.5. Understanding the State	103
7.6. Complete Example	104
7.7. What You Learned	105
 8. Understanding GHZ States	 107
8.1. What is a GHZ State?	107
8.2. Why GHZ States Matter	107
8.3. GHZ vs Bell States	108
8.4. Creating a 3-Qubit GHZ State in QRL	108
8.5. Examining the Quantum State	109
8.6. GHZ Correlations: The Triple Coincidence	111
8.7. Scaling to 4+ Qubits	112
8.8. Partial Measurements on GHZ States	113
8.9. GHZ States and Quantum Non-Locality	115
8.10. Complete Working Example	115
8.11. Performance Note: State Vector Size	117

Table of contents

8.12. What You Learned	118
8.13. Key Concepts	118
8.14. Exercises	119
8.14.1. Exercise 1: Verify 4-Qubit GHZ	119
8.14.2. Exercise 2: Cross-Basis GHZ Measurements	119
8.14.3. Exercise 3: GHZ State Distribution	119
8.14.4. Exercise 4: Sequential Partial Measurements	120
8.14.5. Exercise 5: Large GHZ State	120
8.15. Going Deeper	120
8.15.1. Theory: Why GHZ $\neq$ Bell Pairs	120
8.15.2. Application: Quantum Secret Sharing	121
8.15.3. Connection to MBQC	121
8.16. Summary	122
9. Graph Extraction for MBQC	123
9.1. What is Graph Extraction?	123
9.1.1. Why Graph States Matter	123
9.2. Graph Topologies for Common States	124
9.3. The Graph Extraction API	125
9.3.1. 1. <code>extract_graph()</code> - Convert Relation to Graph	125
9.3.2. 2. <code>analyze_entanglement_structure()</code> - Detect State Type	125
9.3.3. 3. <code>visualize_graph()</code> - Display Graph Structure	126
9.4. Tutorial: Extract Bell State Graph	126
9.5. Tutorial: Extract GHZ State Graph	128
9.6. Scaling to 4+ Qubits	129
9.7. Working with the NetworkX Graph	130
9.8. Understanding the Graph State	131
9.9. GHZ Star Topology Explained	132
9.10. Complete Example: Analyze Multiple States	132
9.11. What You Learned	133
9.12. Key Takeaways	134
9.13. Exercises	134
9.13.1. Exercise 1: Bell State Verification	134

Table of contents

9.13.2. Exercise 2: GHZ Scaling	135
9.13.3. Exercise 3: Graph Properties	135
9.13.4. Exercise 4: W State Graph	135
9.13.5. Exercise 5: Custom Analysis	136
9.14. Next Steps	136
10. Measurement Pattern Generation	137
10.1. What is a Measurement Pattern?	137
10.1.1. Pattern Structure	137
10.2. The Pattern Generation API	138
10.3. Tutorial: Bell State Pattern	138
10.3.1. Understanding the Pattern	139
10.4. Tutorial: GHZ State Pattern	139
10.5. Tutorial: Single-Qubit Gates	140
10.5.1. Hadamard Gate	141
10.5.2. Pauli Gates	142
10.6. Understanding Measurement Angles	142
10.7. Tutorial: Arbitrary Rotations	143
10.8. Tutorial: Two-Qubit Gates	143
10.8.1. CZ Gate (Controlled-Z)	144
10.8.2. CNOT Gate (Controlled-X)	145
10.8.3. CNOT Structure Explained	145
10.8.4. Why CNOT Needs 4 Qubits	147
10.8.5. Universal Quantum Computation	147
10.9. Tutorial: Pattern from Relation	148
10.10 Adaptive Corrections	149
10.11 Pattern Combination	150
10.12 Measurement Depth	150
10.13 Complete Example: Analyze Multiple Patterns	151
10.14 What You Learned	152
10.15 Key Takeaways	152
10.16 Exercises	153
10.16.1 Exercise 1: Pattern Analysis	153
10.16.2 Exercise 2: Measurement Depth	153

10.16.3	Exercise 3: Custom Rotation	154
10.16.4	Exercise 4: GHZ Scaling	154
10.16.5	Exercise 5: Pattern Composer	154
10.16.6	Exercise 6: Pattern Validator	155
10.16.7	Exercise 7: CNOT Analysis	155
10.16.8	Exercise 8: Universal Circuit	155
10.16.9	Exercise 9: CZ vs CNOT	156
10.17	Next Steps	156
11.	Adaptive Corrections and Quantum Teleportation	157
11.1.	Why Adaptive Corrections?	157
11.1.1.	The MBQC Correction Principle	157
11.2.	The Adaptive Corrections API	158
11.3.	Tutorial: Pauli Corrections	158
11.3.1.	X Correction (Bit Flip)	158
11.3.2.	Z Correction (Phase Flip)	159
11.3.3.	Combined XZ Correction	160
11.4.	Quantum Teleportation	160
11.4.1.	The Teleportation Protocol	160
11.4.2.	The Teleportation Pattern	161
11.4.3.	Understanding the Pattern Structure	162
11.5.	Tutorial: Simulating Teleportation	163
11.5.1.	Teleporting Basis States	163
11.5.2.	Teleporting Superposition States	164
11.5.3.	Teleporting Complex States	164
11.6.	The Correction Truth Table	165
11.6.1.	Mathematical Explanation	166
11.7.	Computing Corrections from Patterns	166
11.8.	Multi-Qubit Corrections	167
11.9.	Complete Example: Teleportation Analysis	168
11.10	What You Learned	169
11.11	Key Takeaways	170
11.12	Exercises	171
11.12.1	Exercise 1: Pauli Algebra	171

Table of contents

11.12.2	Exercise 2: Teleportation Statistics	171
11.12.3	Exercise 3: Phase States	171
11.12.4	Exercise 4: Correction Composition	171
11.12.5	Exercise 5: Multi-Round Teleportation	172
11.12.6	Exercise 6: Error Analysis	172
11.12.7	Exercise 7: Custom Protocol	172
11.13	Next Steps	173
 IV. Part IV: Causal Structure as a Programming Primitive		175
12.	Circuit Compilation for MBQC	177
12.1.	From Circuits to Clusters	177
12.1.1.	The Circuit Model vs MBQC	177
12.2.	Building a Circuit Compiler	178
12.2.1.	Step 1: Define a Circuit	178
12.2.2.	Step 2: The Compilation Function	180
12.3.	Tutorial: Compile a Bell State Circuit	182
12.3.1.	Resource Analysis	184
12.4.	Tutorial: GHZ State via Circuit	185
12.5.	The Compilation-Optimization Tradeoff	186
12.5.1.	Naive Compilation	186
12.5.2.	Direct Pattern	186
12.5.3.	When to Use Each Approach	187
12.6.	Tutorial: Quantum Teleportation Circuit	187
12.7.	Circuit Depth Analysis	188
12.8.	Resource Estimation	190
12.9.	Advanced: Parallel Gate Execution	191
12.10	Compilation Strategies	193
12.10.1	Strategy 1: Gate-by-Gate (Current)	193
12.10.2	Strategy 2: Gate Fusion (Advanced)	193
12.10.3	Strategy 3: Pattern Rewriting (Advanced)	194
12.11	What You Learned	194
12.12	Key Takeaways	194

12.13	Exercises	195
12.13.1	Exercise 1: Compile a Swap Gate	195
12.13.2	Exercise 2: Resource Comparison	195
12.13.3	Exercise 3: Parallel Circuit	196
12.13.4	Exercise 4: Rotation Circuit	196
12.13.5	Exercise 5: Teleportation Analysis	196
12.13.6	Exercise 6: Gate Fusion	197
12.13.7	Exercise 7: Depth Optimizer	197
12.13.8	Exercise 8: Resource Predictor	197
12.14	Next Steps	198
13.	The Causal Type Hierarchy	199
13.1.	The Problem: A Program With No Type	200
13.2.	Process Matrices: Quantum Mechanics Without Global Order	201
13.2.1.	Two processes you can write down	203
13.3.	Process Matrices in QRL	203
13.4.	Causal Separability	204
13.4.1.	Causal witnesses	206
13.4.2.	The process that breaks the bound	207
13.5.	QRL's Three Causal Types	209
13.5.1.	PM(n, d) — the general process	209
13.5.2.	Switch(d) — the certified causally nonseparable process	209
13.5.3.	CausalDAG(n) — definite order, made explicit	210
13.5.4.	The hierarchy — and a subtlety	211
13.6.	So What?	212
13.7.	Exercises	213
13.8.	Next Steps	214
14.	The OCB Causal Game	217
14.1.	The Rules	217
14.2.	Why Definite Order Caps the Score at $3/4$	218
14.2.1.	Seeing the bound in QRL	220
14.3.	The Switch Strategy	220
14.4.	Computing It in QRL	221

Table of contents

14.5. The Score Operator	222
14.6. Robustness: How Far Outside the Bound	224
14.7. So What?	225
14.8. Exercises	225
14.9. Next Steps	226
15. Coming Soon	229
 Appendices	 231
A. Installation	231
B. API Reference	233
C. Exercise Solutions	235
C.1. Chapter 1: Quantum Mechanics Basics	235
C.1.1. Solution 1.1: Normalization	235
C.1.2. Solution 1.2: Measurement Probabilities	236
C.1.3. Solution 1.3: Basis Transformation	237
C.1.4. Solution 1.4: Cross-Basis Measurement	237
C.1.5. Solution 1.5: Global Phase	238
C.1.6. Solution 1.6: Bloch Sphere Coordinates	239
C.2. Chapter 2: Entanglement	240
C.2.1. Solution 2.1: Identifying Entanglement	240
C.2.2. Solution 2.2: Schmidt Decomposition	241
C.2.3. Solution 2.3: Entanglement Entropy	242
C.2.4. Solution 2.4: Bell State Transformations	243
C.2.5. Solution 2.5: Partial Measurement	245
C.2.6. Solution 2.6: Creating Bell States	246
C.3. Chapter 3: n-Qubit Systems	247
C.3.1. Solution 3.1: GHZ State Verification	247
C.3.2. Solution 3.2: W State Properties	248
C.3.3. Solution 3.3: Entanglement Entropy Calculation	249

Table of contents

C.3.4.	Solution 3.4: LOCC Impossibility	250
C.3.5.	Solution 3.5: Multipartite Correlation	251
C.3.6.	Solution 3.6: Fragility vs Robustness	253
C.3.7.	Solution 3.7: n-Qubit Generalization	255
C.4.	Chapter 4: MBQC Foundations	256
C.4.1.	Solution 4.1: Cluster State Preparation	256
C.4.2.	Solution 4.2: Measurement Outcomes	258
C.4.3.	Solution 4.3: Gate Teleportation	259
C.4.4.	Solution 4.4: Pattern Design	261
C.4.5.	Solution 4.5: Resource Scaling	262
C.4.6.	Solution 4.6: Advantage Analysis	264
C.5.	Chapter 5: Cluster States	265
C.5.1.	Solution 5.1: Stabilizer Verification	265
C.5.2.	Solution 5.2: 4-Qubit Chain Stabilizers	267
C.5.3.	Solution 5.3: Stabilizer Commutation	268
C.5.4.	Solution 5.4: Local Complementation	269
C.5.5.	Solution 5.5: Measurement Update	270
C.5.6.	Solution 5.6: Graph State Entanglement	272
C.5.7.	Solution 5.7: Resource Comparison	273
C.6.	Chapter 6: Measurement Patterns	274
C.6.1.	Solution 6.1: Hadamard Compilation	274
C.6.2.	Solution 6.2: Rotation Angle	276
C.6.3.	Solution 6.3: CNOT Pattern	277
C.6.4.	Solution 6.4: Adaptive Measurement	278
C.6.5.	Solution 6.5: Pattern Optimization	280
C.6.6.	Solution 6.6: Deterministic Gates	281
C.6.7.	Solution 6.7: Circuit to Pattern	282

Welcome

This book has one destination: understanding why the **quantum switch has a type**.

That sentence will not make sense yet. By Chapter 15, it will be the most natural thing in the world.

The idea in one paragraph

Every quantum programming language you have heard of — Qiskit, Cirq, Q#, QWIRE, even the most mathematically sophisticated typed quantum languages — shares one hidden assumption: operations happen in a fixed order. A comes before B, or B comes before A, and this is decided before the program runs. Quantum mechanics does not require this. There exist physical processes in which the causal order of operations is itself in quantum superposition — genuinely indefinite, not just unknown. No existing language can represent these processes, because none have a *type* for causal structure. QRL does.

The central result of this book is the **QuantumSwitch Soundness Theorem**: any well-typed `switch(f, g, c)` expression in QRL denotes a *causally nonseparable* process matrix. The type system statically guarantees a property of physical reality. That is what types are for.

Welcome

What you will learn

This book builds the result from scratch, in four parts:

Part I — Quantum Foundations (Chapters 1–3) Qubits, entanglement, and n-qubit systems. The mathematical toolkit: state vectors, density matrices, tensor products, measurement. You need this before everything else.

Part II — Measurement-Based Quantum Computing (Chapters 4–6) Cluster states, graph states, measurement patterns. This is how QRL compiles to photonic hardware. It is also the layer that sits *below* causal structure — understanding it makes the causal layer’s position in the architecture clear.

Part III — QRL Programming (Chapters 7–12) Hands-on: Bell states, GHZ states, graph extraction, pattern generation, adaptive corrections, circuit compilation. You will write real QRL programs and run them.

Part IV — Causal Structure as a Programming Primitive (Chapters 13–15) The destination. Process matrices, the OCB causal game, the type hierarchy, and finally the Soundness Theorem with a full proof walkthrough. By Chapter 15 you will understand not just *what* the theorem says but *why* it holds and why no other language can even state it.

Prerequisites

- **Mathematics:** Linear algebra (vectors, matrices, eigenvalues), complex numbers, basic probability
- **Programming:** Python basics — QRL is implemented in Python
- **Physics:** Helpful but not required — quantum mechanics is developed from scratch in Part I

How to use this book

If you want the full arc: read linearly. Each part builds on the previous one.

If you already know quantum mechanics: start at Chapter 4 (MBQC Foundations) or Chapter 7 (first QRL program).

If you want to go straight to the result: read the paper first ([arXiv](#)), then use this book to understand the implementation and the mathematics behind the theorem.

The code

All code examples run against the QRL implementation:

```
pip install qrl
```

The implementation is ~13,600 lines of Python with 896 passing tests. Everything shown in this book is tested against that implementation.

Source: github.com/entangledcode/qrl

Welcome

About the author

David Coldeira is an independent researcher and scientific software engineer with a BSc in Physics. QRL is his primary research project — a quantum programming language in which causal structure is a first-class type-level primitive.

- Email: david@entangledcode.dev
 - Website: entangledcode.dev
 - GitHub: [dcoldeira](https://github.com/dcoldeira)
-

License

© 2026 David Coldeira. The text of this book is licensed under [Creative Commons Attribution-NonCommercial-ShareAlike 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/) (CC BY-NC-SA 4.0). Code examples are licensed under the MIT License.

Let's begin.

Part I.

Part I: Quantum Foundations

1. Quantum Mechanics Basics

Welcome to the quantum world. In this chapter, we'll build the mathematical foundation you need to understand quantum computing—from first principles, with rigorous proofs and no hand-waving.

1.1. The Quantum State

In classical physics, a bit is either 0 or 1. In quantum mechanics, a **qubit** (quantum bit) can exist in a **superposition** of both states simultaneously.

1.1.1. Mathematical Representation

A quantum state is represented as a **vector** in a complex vector space called a **Hilbert space**. For a single qubit, this is a 2-dimensional space.

The computational basis states are:

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

Notation: We use **Dirac notation** (or “bra-ket” notation):

- $|0\rangle$ is pronounced “ket zero” - represents a column vector
- $\langle 0|$ is pronounced “bra zero” - represents a row vector (conjugate transpose)

1. Quantum Mechanics Basics

1.1.2. General Qubit State

The most general state of a qubit is a **superposition**:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

where $\alpha, \beta \in \mathbb{C}$ (complex numbers) and $|\alpha|^2 + |\beta|^2 = 1$.

Key insight: The state exists in *both* $|0\rangle$ and $|1\rangle$ simultaneously until measured.

1.2. The Superposition Principle

One of the foundational principles of quantum mechanics:

Superposition Principle

If $|\psi_1\rangle$ and $|\psi_2\rangle$ are valid quantum states, then any linear combination:

$$|\psi\rangle = \alpha|\psi_1\rangle + \beta|\psi_2\rangle$$

is also a valid quantum state (provided it's normalized: $|\alpha|^2 + |\beta|^2 = 1$).

1.2.1. Example: The Hadamard State

One of the most important quantum states is:

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

This is an **equal superposition** of $|0\rangle$ and $|1\rangle$.

1.3. Measurement and the Born Rule

Similarly, we can define:

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Together, $\{|+\rangle, |-\rangle\}$ form an alternative **basis** for the qubit's Hilbert space (the **X-basis** or **Hadamard basis**).

1.3. Measurement and the Born Rule

When we measure a quantum state, something dramatic happens: the superposition **collapses** to one of the basis states.

1.3.1. The Born Rule

Born Rule (Measurement Postulate)

Given a quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, when measured in the computational (Z) basis:

- Probability of measuring $|0\rangle$: $P(0) = |\alpha|^2$
- Probability of measuring $|1\rangle$: $P(1) = |\beta|^2$

After measurement, the state **collapses** to the measured outcome.

1.3.2. Example: Measuring $|+\rangle$

Consider the state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

When measured in the Z-basis:

- $P(0) = \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$

1. Quantum Mechanics Basics

- $P(1) = \left| \frac{1}{\sqrt{2}} \right|^2 = \frac{1}{2}$

Interpretation: We get 0 or 1 with equal probability (50/50), and the state becomes either $|0\rangle$ or $|1\rangle$ after measurement.

1.4. Normalization Condition

The probabilities must sum to 1. This gives us the **normalization condition**:

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

Let's prove this is required.

Proof: Normalization Ensures Valid Probabilities

Claim: For a quantum state to yield valid probabilities, it must satisfy $|\alpha|^2 + |\beta|^2 = 1$.

Proof:

By the axioms of probability, the sum of all possible outcomes must equal 1:

$$P(0) + P(1) = 1$$

By the Born rule:

$$P(0) = |\alpha|^2, \quad P(1) = |\beta|^2$$

Therefore:

$$|\alpha|^2 + |\beta|^2 = 1 \quad \square$$

This is called the **normalization condition**. Any valid quantum state must be normalized.

1.5. Measurement in Different Bases

1.4.1. Inner Product and Normalization

We can express normalization using the **inner product** (or dot product) in Hilbert space:

$$\langle \psi | \psi \rangle = 1$$

where $\langle \psi | = (\alpha^* \quad \beta^*)$ is the conjugate transpose of $|\psi\rangle$.

Calculation:

$$\langle \psi | \psi \rangle = (\alpha^* \quad \beta^*) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha^* \alpha + \beta^* \beta = |\alpha|^2 + |\beta|^2 = 1$$

1.5. Measurement in Different Bases

The Born rule applies to **any** orthonormal basis, not just the computational basis.

1.5.1. Z-Basis vs X-Basis

Z-basis (computational basis):

$$\{|0\rangle, |1\rangle\}$$

X-basis (Hadamard basis):

$$\{|+\rangle, |-\rangle\} = \left\{ \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \right\}$$

1. Quantum Mechanics Basics

1.5.2. Example: Measuring $|0\rangle$ in X-Basis

To measure $|0\rangle$ in the X-basis, we first **express it in that basis**:

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

Verification:

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

$$= \frac{1}{2}(|0\rangle + |1\rangle) + \frac{1}{2}(|0\rangle - |1\rangle) = \frac{1}{2}(2|0\rangle) = |0\rangle \quad \checkmark$$

Now applying Born rule:

- $P(+)=\left|\frac{1}{\sqrt{2}}\right|^2=\frac{1}{2}$
- $P(-)=\left|\frac{1}{\sqrt{2}}\right|^2=\frac{1}{2}$

Key insight: The state $|0\rangle$ is a **superposition** when viewed in the X-basis! This is called **basis-dependent measurement** or **contextuality**.

1.6. Global Phase and Physical Equivalence

Consider two states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$|\psi'\rangle = e^{i\theta}(\alpha|0\rangle + \beta|1\rangle) = e^{i\theta}|\psi\rangle$$

1.6. Global Phase and Physical Equivalence

where $\theta \in \mathbb{R}$ and $e^{i\theta}$ is a **global phase factor**.

Claim: These states are **physically indistinguishable**.

Why? Measurement probabilities are unchanged:

$$P(0) = |e^{i\theta}\alpha|^2 = |e^{i\theta}|^2|\alpha|^2 = |\alpha|^2$$

since $|e^{i\theta}| = 1$ for any real θ .

Global Phase Irrelevance

States that differ only by a global phase factor are **physically equivalent**:

$$|\psi\rangle \equiv e^{i\theta}|\psi\rangle$$

However, **relative phases** (phases between different components) are observable and physically meaningful.

1.6.1. Example: Relative Phase Matters

These two states are **different**:

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

The relative phase between $|0\rangle$ and $|1\rangle$ components (+1 vs -1) is observable when measured in the X-basis!

1. Quantum Mechanics Basics

1.7. Quantum State Space: The Bloch Sphere

For a single qubit, we can visualize the state space geometrically.

Any single-qubit state can be written as:

$$|\psi\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle$$

where $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi)$.

This parameterization maps quantum states to points on a **unit sphere** in 3D space called the **Bloch sphere**.

Key points on the Bloch sphere:

- **North pole** ($\theta = 0$): $|0\rangle$
- **South pole** ($\theta = \pi$): $|1\rangle$
- **Equator** ($\theta = \pi/2$):
 - $\phi = 0$: $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
 - $\phi = \pi$: $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
 - $\phi = \pi/2$: $|+i\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$
 - $\phi = 3\pi/2$: $|-i\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$

Why the Bloch Sphere Matters

The Bloch sphere gives us geometric intuition about quantum states. Antipodal points (opposite ends of a diameter) represent **orthogonal states** - states that are maximally distinguishable by measurement.

1.8. Pure States vs Mixed States

So far we've discussed **pure states** - states with definite (though possibly superposed) quantum information.

In reality, quantum systems often exist in **mixed states** due to:

- Incomplete knowledge about the system
- Interaction with the environment (decoherence)
- Entanglement with other systems

Mixed states are described by **density matrices** (we'll cover this in later chapters).

1.9. Summary

In this chapter, you learned:

✓ Quantum states are vectors in complex Hilbert space ✓ Superposition: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ ✓ Born rule: Measurement probabilities $P(i) = |\alpha_i|^2$ ✓ Normalization: $|\alpha|^2 + |\beta|^2 = 1$ ensures valid probabilities ✓ Basis-dependent measurement: States look different in different bases ✓ Global phase is irrelevant, relative phase is observable ✓ Bloch sphere: Geometric representation of qubit states

1.10. Exercises

1.10.1. Exercise 1.1: Normalization

Given the (unnormalized) state:

$$|\psi\rangle = 2|0\rangle + 2i|1\rangle$$

1. Quantum Mechanics Basics

- (a) Calculate $\langle\psi|\psi\rangle$.
- (b) Normalize the state to obtain $|\psi_{\text{norm}}\rangle$.
- (c) Verify that $\langle\psi_{\text{norm}}|\psi_{\text{norm}}\rangle = 1$.

Solution $\rightarrow$

1.10.2. Exercise 1.2: Measurement Probabilities

Consider the state:

$$|\psi\rangle = \frac{1}{2}|0\rangle + \frac{\sqrt{3}}{2}|1\rangle$$

- (a) Calculate $P(0)$ and $P(1)$ when measured in the Z-basis.
- (b) Verify that $P(0) + P(1) = 1$.
- (c) After measuring and obtaining outcome 1, what is the post-measurement state?

Solution $\rightarrow$

1.10.3. Exercise 1.3: Basis Transformation

Express the state $|1\rangle$ in the X-basis (i.e., as a linear combination of $|+\rangle$ and $|-\rangle$).

Hint: Use the inverse relations:

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

Solution $\rightarrow$

1.10.4. Exercise 1.4: Cross-Basis Measurement

Consider the state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

- (a) What are $P(0)$ and $P(1)$ when measured in the Z-basis?
- (b) What are $P(+)$ and $P(-)$ when measured in the X-basis?
- (c) Explain why the probabilities differ between the two bases.

Solution →

1.10.5. Exercise 1.5: Global Phase

Show that the states $|\psi\rangle = |0\rangle$ and $|\psi'\rangle = -|0\rangle = e^{i\pi}|0\rangle$ are physically equivalent by proving they yield identical measurement probabilities in both Z-basis and X-basis.

Solution →

1.10.6. Exercise 1.6: Bloch Sphere Coordinates

A qubit is in the state:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{e^{i\pi/4}}{\sqrt{2}}|1\rangle$$

Determine the Bloch sphere angles θ and ϕ for this state.

Solution →

Next: [Chapter 2: Entanglement](#) - Where quantum mechanics gets truly strange

2. Entanglement

Entanglement is the most profoundly non-classical phenomenon in quantum mechanics. Einstein called it “spooky action at a distance.” It’s the foundation of quantum computing’s power—and the core primitive in QRL.

2.1. Two-Qubit Systems

Before diving into entanglement, we need to understand how to describe multiple qubits mathematically.

2.1.1. Tensor Product

When we have two independent quantum systems, we combine them using the **tensor product** (denoted $\otimes$).

Example: System A in state $|0\rangle_A$ and system B in state $|1\rangle_B$ combine to:

$$|0\rangle_A \otimes |1\rangle_B = |01\rangle$$

Shorthand notation: We often write $|01\rangle$ instead of $|0\rangle_A \otimes |1\rangle_B$.

2. Entanglement

2.1.2. Computational Basis for Two Qubits

The 4 basis states for a 2-qubit system:

$$|00\rangle, \quad |01\rangle, \quad |10\rangle, \quad |11\rangle$$

As vectors in $\mathbb{C}^4$:

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

2.1.3. General Two-Qubit State

The most general 2-qubit state is:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$

with normalization: $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$.

2.2. Separable vs Entangled States

Not all 2-qubit states are created equal. There's a fundamental distinction:

2.2. Separable vs Entangled States

2.2.1. Separable States

A state is **separable** (or **product state**) if it can be written as:

$$|\psi\rangle = |\phi\rangle_A \otimes |\chi\rangle_B$$

where $|\phi\rangle_A$ is a single-qubit state for system A and $|\chi\rangle_B$ is a single-qubit state for system B.

Example:

$$|\psi\rangle = |0\rangle_A \otimes |+\rangle_B = |0\rangle_A \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)_B = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)$$

This state is **separable** - the two qubits are independent.

2.2.2. Entangled States

A state is **entangled** if it **cannot** be written as a tensor product of single-qubit states.

Definition: Entanglement

A 2-qubit state $|\psi\rangle$ is **entangled** if and only if it cannot be expressed as:

$$|\psi\rangle = |\phi\rangle_A \otimes |\chi\rangle_B$$

for any single-qubit states $|\phi\rangle_A$ and $|\chi\rangle_B$.

2. Entanglement

2.3. Bell States: Maximally Entangled Pairs

The four **Bell states** are the most important entangled states in quantum computing:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$$

$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$$

$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$$

2.3.1. Why Can't Bell States Be Factored?

Let's prove that $|\Phi^+\rangle$ cannot be written as a product state.

Proof: $|\Phi^+\rangle$ Is Entangled

Claim: $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ cannot be written as $|\phi\rangle_A \otimes |\chi\rangle_B$.

Proof by contradiction:

Assume $|\Phi^+\rangle = |\phi\rangle_A \otimes |\chi\rangle_B$ where:

$$|\phi\rangle_A = \alpha|0\rangle + \beta|1\rangle, \quad |\chi\rangle_B = \gamma|0\rangle + \delta|1\rangle$$

Then:

$$|\phi\rangle_A \otimes |\chi\rangle_B = (\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle)$$

2.4. Physical Meaning of Entanglement

$$= \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle$$

Comparing with $|\Phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + 0|01\rangle + 0|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$:

$$\alpha\gamma = \frac{1}{\sqrt{2}}, \quad \alpha\delta = 0, \quad \beta\gamma = 0, \quad \beta\delta = \frac{1}{\sqrt{2}}$$

From $\alpha\delta = 0$: either $\alpha = 0$ or $\delta = 0$.

- If $\alpha = 0$: then $\alpha\gamma = 0 \neq \frac{1}{\sqrt{2}} \not\prec$ contradiction
- If $\delta = 0$: then $\beta\delta = 0 \neq \frac{1}{\sqrt{2}} \not\prec$ contradiction

Therefore, $|\Phi^+\rangle$ cannot be factored. $\square$

2.4. Physical Meaning of Entanglement

When two qubits are entangled in the state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$:

1. **Before measurement:** Neither qubit has a definite state. They exist in a **correlated superposition**.
2. **Measuring qubit A:**
 - 50% chance of measuring 0 $\rightarrow$ qubit B **instantly** becomes $|0\rangle$
 - 50% chance of measuring 1 $\rightarrow$ qubit B **instantly** becomes $|1\rangle$
3. **Perfect correlation:** The measurement outcomes are perfectly correlated, *even if the qubits are separated by light-years*.

This is what troubled Einstein - the “spooky action at a distance.” However, no information travels faster than light (you can’t control which outcome you get).

2. Entanglement

2.5. Schmidt Decomposition

Every bipartite pure state can be expressed in a special canonical form.

Schmidt Decomposition Theorem

Any pure state $|\psi\rangle$ of a bipartite system A-B can be written as:

$$|\psi\rangle = \sum_{i=1}^r \lambda_i |u_i\rangle_A \otimes |v_i\rangle_B$$

where:

- $\{|u_i\rangle_A\}$ is an orthonormal basis for system A
- $\{|v_i\rangle_B\}$ is an orthonormal basis for system B
- $\lambda_i > 0$ are real numbers (Schmidt coefficients) with $\sum_i \lambda_i^2 = 1$
- r is the **Schmidt rank** (number of non-zero terms)

This decomposition is unique (up to reordering and degeneracies).

2.5.1. Schmidt Rank and Entanglement

The Schmidt rank tells us about entanglement:

- $r = 1$: State is separable (product state)
- $r > 1$: State is entangled
- $r = \min(\dim A, \dim B)$: State is **maximally entangled**

2.5.2. Example: Schmidt Decomposition of $|\Phi^+\rangle$

The Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is already in Schmidt form:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}|0\rangle_A \otimes |0\rangle_B + \frac{1}{\sqrt{2}}|1\rangle_A \otimes |1\rangle_B$$

2.6. Entanglement Entropy

Schmidt coefficients: $\lambda_1 = \lambda_2 = \frac{1}{\sqrt{2}}$

Schmidt rank: $r = 2$ (maximal for 2 qubits)

2.5.3. Example: Separable State

Consider $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle) = |0\rangle_A \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)_B$.

Schmidt decomposition:

$$|\psi\rangle = 1 \cdot |0\rangle_A \otimes |+\rangle_B$$

Schmidt rank: $r = 1$ (separable)

2.6. Entanglement Entropy

How do we quantify *how much* entanglement a state has?

2.6.1. Reduced Density Matrix

For a bipartite state $|\psi\rangle_{AB}$, the **reduced density matrix** for system A is:

$$\rho_A = \text{Tr}_B(|\psi\rangle\langle\psi|)$$

where Tr_B means “trace out system B” (sum over all states of B).

For a state in Schmidt form $|\psi\rangle = \sum_i \lambda_i |u_i\rangle_A \otimes |v_i\rangle_B$:

$$\rho_A = \sum_i \lambda_i^2 |u_i\rangle\langle u_i|$$

2. Entanglement

The eigenvalues of ρ_A are the **Schmidt coefficients squared**: $\{\lambda_1^2, \lambda_2^2, \dots\}$.

2.6.2. Von Neumann Entropy

The **entanglement entropy** is the von Neumann entropy of the reduced density matrix:

$$S(\rho_A) = -\text{Tr}(\rho_A \log_2 \rho_A) = -\sum_i \lambda_i^2 \log_2(\lambda_i^2)$$

Properties of Entanglement Entropy

- $S = 0$: State is separable (no entanglement)
- $S > 0$: State is entangled
- $S = \log_2 d$: State is maximally entangled (for d -dimensional subsystems)
- **For 2 qubits:** $S_{\max} = \log_2 2 = 1$ bit

2.6.3. Example: Entropy of $|\Phi^+\rangle$

For $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$:

Schmidt coefficients: $\lambda_1 = \lambda_2 = \frac{1}{\sqrt{2}}$

Eigenvalues of ρ_A : $\frac{1}{2}, \frac{1}{2}$

Entropy:

$$S = -\frac{1}{2} \log_2 \left(\frac{1}{2} \right) - \frac{1}{2} \log_2 \left(\frac{1}{2} \right) = -\frac{1}{2}(-1) - \frac{1}{2}(-1) = 1$$

2.6. Entanglement Entropy

Result: $S = 1$ bit - maximally entangled! ✓

Proof: Bell States Have Maximal Entropy

Claim: All four Bell states have entanglement entropy $S = 1$ bit.

Proof:

All Bell states have Schmidt decomposition with coefficients $\lambda_1 = \lambda_2 = \frac{1}{\sqrt{2}}$.

For example:

- $|\Phi^+\rangle = \frac{1}{\sqrt{2}}|0\rangle|0\rangle + \frac{1}{\sqrt{2}}|1\rangle|1\rangle$
- $|\Psi^+\rangle = \frac{1}{\sqrt{2}}|0\rangle|1\rangle + \frac{1}{\sqrt{2}}|1\rangle|0\rangle$

For all: $\lambda_1^2 = \lambda_2^2 = \frac{1}{2}$

Therefore:

$$S = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1 \text{ bit} \quad \square$$

2.6.4. Example: Partially Entangled State

Consider:

$$|\psi\rangle = \frac{\sqrt{3}}{2}|00\rangle + \frac{1}{2}|11\rangle$$

Schmidt coefficients: $\lambda_1 = \frac{\sqrt{3}}{2}, \lambda_2 = \frac{1}{2}$

Entropy:

$$S = -\frac{3}{4} \log_2 \frac{3}{4} - \frac{1}{4} \log_2 \frac{1}{4} \approx 0.811 \text{ bits}$$

This is **partially entangled** ($0 < S < 1$).

2. Entanglement

2.7. Bell's Theorem and Non-Locality

Entanglement leads to correlations that cannot be explained by any local hidden variable theory.

2.7.1. EPR Paradox

In 1935, Einstein, Podolsky, and Rosen (EPR) argued that quantum mechanics must be incomplete. They believed:

1. Elements of reality exist for unmeasured properties
2. Locality: Measuring one particle can't instantly affect a distant particle

These assumptions lead to **local hidden variable theories**.

2.7.2. Bell's Inequality

In 1964, John Bell proved that local hidden variable theories predict:

$$|E(a, b) - E(a, c)| \leq 1 + E(b, c)$$

where $E(x, y)$ is the correlation between measurements in directions x and y .

Quantum mechanics violates this inequality!

For Bell states measured in appropriate bases:

$$|E(a, b) - E(a, c)| = 2\sqrt{2} \approx 2.83 > 1 + E(b, c)$$

This has been experimentally verified countless times. Nature is **non-local** - entanglement is real.

Implications for Quantum Computing

Bell's theorem proves that quantum correlations are fundamentally different from classical correlations. This is the source of quantum computing's power:

- Quantum algorithms exploit entanglement for exponential speedups
- Classical computers cannot efficiently simulate entangled systems
- QRL makes entanglement a first-class primitive to harness this power

2.8. Measurement of Entangled States

When we measure one qubit of an entangled pair, the entire state collapses.

2.8.1. Example: Measuring $|\Phi^+\rangle$

Start with: $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$

Measure qubit A in Z-basis:

- **Outcome 0** (probability $\frac{1}{2}$):
 - State collapses to $|00\rangle$
 - Qubit B is now in state $|0\rangle$
- **Outcome 1** (probability $\frac{1}{2}$):
 - State collapses to $|11\rangle$
 - Qubit B is now in state $|1\rangle$

2. Entanglement

Key observation: Measuring qubit A **instantaneously** determines the state of qubit B, regardless of distance!

2.8.2. Partial Measurement Formula

For a general state $|\psi\rangle = \sum_{ij} c_{ij} |i\rangle_A |j\rangle_B$:

Measuring qubit A and getting outcome i :

$$|\psi\rangle \xrightarrow{\text{measure } A=i} \frac{1}{\sqrt{P(i)}} \sum_j c_{ij} |i\rangle_A |j\rangle_B = |i\rangle_A \otimes \left(\frac{1}{\sqrt{P(i)}} \sum_j c_{ij} |j\rangle_B \right)$$

where $P(i) = \sum_j |c_{ij}|^2$ is the probability of outcome i .

2.9. Summary

In this chapter, you learned:

✓ Tensor product combines multiple quantum systems ✓ Separable states can be factored: $|\phi\rangle_A \otimes |\chi\rangle_B$ ✓ Entangled states cannot be factored ✓ Bell states are maximally entangled 2-qubit states ✓ Schmidt decomposition is the canonical form for bipartite states ✓ Entanglement entropy quantifies entanglement: $S = -\sum_i \lambda_i^2 \log_2 \lambda_i^2$ ✓ Bell's theorem proves quantum non-locality ✓ Measuring one qubit collapses the entangled partner

2.10. Exercises

2.10.1. Exercise 2.1: Identifying Entanglement

Which of the following states are entangled? Prove your answer.

(a) $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)$

(b) $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$

(c) $|\psi\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$

Solution $\rightarrow$

2.10.2. Exercise 2.2: Schmidt Decomposition

Find the Schmidt decomposition for:

$$|\psi\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$$

Calculate the Schmidt rank and entanglement entropy.

Solution $\rightarrow$

2.10.3. Exercise 2.3: Entanglement Entropy

Calculate the entanglement entropy for:

$$|\psi\rangle = \frac{2}{\sqrt{5}}|00\rangle + \frac{1}{\sqrt{5}}|11\rangle$$

Is this state maximally entangled?

Solution $\rightarrow$

2. Entanglement

2.10.4. Exercise 2.4: Bell State Transformations

Prove that all four Bell states can be transformed into each other using single-qubit gates (Pauli gates):

$$\begin{aligned} X|0\rangle &= |1\rangle, & X|1\rangle &= |0\rangle \\ Z|0\rangle &= |0\rangle, & Z|1\rangle &= -|1\rangle \end{aligned}$$

Hint: Start with $|\Phi^+\rangle$ and apply X or Z gates to derive the others.

Solution $\rightarrow$

2.10.5. Exercise 2.5: Partial Measurement

Given the state $|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$:

(a) If qubit A is measured in the Z-basis and outcome 0 is obtained, what is the post-measurement state?

(b) What is the probability of measuring qubit B as 1 after the measurement in part (a)?

Solution $\rightarrow$

2.10.6. Exercise 2.6: Creating Bell States

In a quantum circuit, a Bell state $|\Phi^+\rangle$ is created from $|00\rangle$ using:

1. Hadamard gate on qubit A: $H|0\rangle_A = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)_A$
2. CNOT gate with A as control and B as target

Verify that this produces $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

Solution $\rightarrow$

Next: [Chapter 3: n-Qubit Systems](#) - Scaling to many qubits

3. n-Qubit Systems

Moving beyond 2 qubits unlocks the full richness of quantum entanglement. With 3 or more qubits, we encounter genuinely different classes of entanglement—states that cannot be converted to each other using local operations.

3.1. Tensor Products for n Qubits

3.1.1. The n-Qubit Hilbert Space

An n -qubit system lives in a Hilbert space of dimension 2^n .

Computational basis: $\{|b_1 b_2 \dots b_n\rangle : b_i \in \{0, 1\}\}$

For 3 qubits: $|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle$ (8 states)

For n qubits: 2^n basis states

3.1.2. General n-Qubit State

$$|\psi\rangle = \sum_{b_1, \dots, b_n \in \{0, 1\}} c_{b_1 \dots b_n} |b_1 \dots b_n\rangle$$

with normalization: $\sum_{b_1, \dots, b_n} |c_{b_1 \dots b_n}|^2 = 1$

Note: The number of complex amplitudes grows exponentially: 2^n complex numbers!

3. *n*-Qubit Systems

- 10 qubits: 1,024 amplitudes
- 20 qubits: 1,048,576 amplitudes
- 50 qubits: $\sim 10^{15}$ amplitudes (more than can be stored classically!)

This exponential growth is both the **power** and the **challenge** of quantum computing.

3.2. GHZ States: Maximal Multipartite Entanglement

3.2.1. Definition

The **GHZ state** (Greenberger-Horne-Zeilinger) for n qubits is:

$$|\text{GHZ}_n\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}) = \frac{1}{\sqrt{2}}(|00 \dots 0\rangle + |11 \dots 1\rangle)$$

Examples:

- $n = 2$: $|\text{GHZ}_2\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = |\Phi^+\rangle$ (Bell state)
- $n = 3$: $|\text{GHZ}_3\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$
- $n = 4$: $|\text{GHZ}_4\rangle = \frac{1}{\sqrt{2}}(|0000\rangle + |1111\rangle)$

3.2.2. Properties of GHZ States

1. **Perfectly correlated:** Measuring any qubit in the Z-basis gives 0 or 1, and all other qubits will have the same value.
2. **Fragile:** Measuring even one qubit completely destroys the multipartite entanglement, leaving a product state.
3. **Maximal bipartite entanglement:** For any bipartition (splitting qubits into two groups), the entanglement entropy is maximal.

3.2. GHZ States: Maximal Multipartite Entanglement

3.2.3. GHZ Correlations

For $|\text{GHZ}_3\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$:

Measuring all qubits in Z-basis:

- $P(000) = \frac{1}{2}$ - all measure 0
- $P(111) = \frac{1}{2}$ - all measure 1
- $P(\text{any other outcome}) = 0$ - impossible!

This perfect 3-way correlation has no classical analogue.

3.2.4. Entanglement Entropy of GHZ

For a bipartition $A|B$ where A has k qubits and B has $n - k$ qubits:

$$S_{A|B} = 1 \text{ bit}$$

regardless of how we partition! (as long as both groups are non-empty)

Proof: GHZ Has Maximal Bipartite Entropy

Claim: For $|\text{GHZ}_n\rangle$, any bipartition yields $S = 1$ bit.

Proof:

Consider partition into group A (first k qubits) and group B (remaining $n - k$ qubits).

The GHZ state can be written:

$$|\text{GHZ}_n\rangle = \frac{1}{\sqrt{2}}|0\rangle_A^{\otimes k}|0\rangle_B^{\otimes(n-k)} + \frac{1}{\sqrt{2}}|1\rangle_A^{\otimes k}|1\rangle_B^{\otimes(n-k)}$$

This is in Schmidt form with coefficients $\lambda_1 = \lambda_2 = \frac{1}{\sqrt{2}}$.

Therefore:

3. *n*-Qubit Systems

$$S = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1 \text{ bit} \quad \square$$

3.3. W States: Robust Entanglement

3.3.1. Definition

The **W state** for n qubits is an equal superposition of all basis states with exactly one qubit in $|1\rangle$:

$$|W_n\rangle = \frac{1}{\sqrt{n}}(|10\cdots 0\rangle + |01\cdots 0\rangle + \cdots + |0\cdots 01\rangle)$$

Examples:

- $n = 3$: $|W_3\rangle = \frac{1}{\sqrt{3}}(|100\rangle + |010\rangle + |001\rangle)$
- $n = 4$: $|W_4\rangle = \frac{1}{2}(|1000\rangle + |0100\rangle + |0010\rangle + |0001\rangle)$

3.3.2. Properties of W States

1. **Robust:** If you measure one qubit and get 0, the remaining qubits are still entangled in a W state with $n - 1$ qubits.
2. **Partial bipartite entanglement:** For bipartition $1|23\cdots n$, the entanglement is less than maximal.
3. **Symmetric:** All qubits play equivalent roles (state is unchanged by permuting qubits).

3.3. *W States: Robust Entanglement*

3.3.3. **W State Resilience**

For $|W_3\rangle = \frac{1}{\sqrt{3}}(|100\rangle + |010\rangle + |001\rangle)$:

Measuring qubit 1 in Z-basis:

- **Outcome 1** (probability $\frac{1}{3}$):
 - State collapses to $|100\rangle$ (separable)
- **Outcome 0** (probability $\frac{2}{3}$):
 - State collapses to $\frac{1}{\sqrt{2}}(|010\rangle + |001\rangle) = |0\rangle \otimes \frac{1}{\sqrt{2}}(|10\rangle + |01\rangle)$
 - Qubits 2 and 3 are still entangled in a Bell state!

Key difference from GHZ: W states preserve entanglement when losing qubits.

3.3.4. **Entanglement Entropy of W**

For $|W_3\rangle$ with bipartition 1|23:

The state can be written:

$$|W_3\rangle = \frac{1}{\sqrt{3}}|1\rangle|00\rangle + \frac{\sqrt{2}}{\sqrt{3}}|0\rangle \cdot \frac{1}{\sqrt{2}}(|10\rangle + |01\rangle)$$

Schmidt coefficients: $\lambda_1 = \frac{1}{\sqrt{3}}, \lambda_2 = \sqrt{\frac{2}{3}}$

Entropy:

$$S = -\frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3} \approx 0.918 \text{ bits}$$

This is less than 1 bit - W states are not maximally entangled for bipartitions!

3. *n*-Qubit Systems

3.4. GHZ vs W: Different Entanglement Classes

GHZ and W states represent fundamentally different types of entanglement.

3.4.1. LOCC Equivalence

Two states are **LOCC equivalent** (Local Operations and Classical Communication) if one can be transformed into the other using:

- Local quantum operations on individual qubits
- Classical communication between parties

Theorem: GHZ and W Are LOCC-Inequivalent

For $n \geq 3$ qubits, GHZ and W states belong to **different entanglement classes**. You cannot convert:

$$|\text{GHZ}_n\rangle \xrightarrow{\text{LOCC}} |\text{W}_n\rangle$$

$$|\text{W}_n\rangle \xrightarrow{\text{LOCC}} |\text{GHZ}_n\rangle$$

These represent genuinely different types of multipartite entanglement.

Physical interpretation:

- **GHZ:** Fragile, maximal correlations, “all-or-nothing” entanglement
- **W:** Robust, partial correlations, “distributed” entanglement

3.4.2. Summary of Differences

3.5. Multipartite Entanglement: Beyond 2 Parties

Property	GHZ State	W State
Form	$\frac{1}{\sqrt{2}}(00 \dots 0\rangle + 11 \dots 1\rangle)$	$\frac{1}{\sqrt{n}} \sum_i 0 \dots 010 \dots 0\rangle_i$
Bipartite entropy	$S = 1$ (maximal)	$S < 1$ (sub-maximal)
Robustness	Fragile to loss	Robust to qubit loss
Correlations	Perfect all-way	Partial pairwise
LOCC class	Distinct	Distinct

3.5. Multipartite Entanglement: Beyond 2 Parties

3.5.1. Genuine Multipartite Entanglement

A state has **genuine multipartite entanglement** if it is entangled across all possible bipartitions.

Example: $|\text{GHZ}_3\rangle$ is genuinely tripartite entangled:

- Bipartition 1|23: entangled ✓
- Bipartition 2|13: entangled ✓
- Bipartition 3|12: entangled ✓

3.5.2. Biseparable States

A state is **biseparable** if it can be written as a mixture of states that are separable across some bipartition.

Example: $|\psi\rangle = |0\rangle_1 \otimes |\Phi^+\rangle_{23}$ is biseparable (qubit 1 is separable from qubits 2-3).

This has entanglement but not genuine tripartite entanglement.

3. *n*-Qubit Systems

3.6. Measuring n-Qubit States

3.6.1. Partial Measurements

When measuring k qubits of an n -qubit state, the remaining $n - k$ qubits collapse to a state determined by the measurement outcome.

General formula: For state $|\psi\rangle = \sum_{i,j} c_{ij} |i\rangle_A |j\rangle_B$ where A represents the measured qubits and B the unmeasured:

Measuring A and getting outcome i :

$$|\psi\rangle \xrightarrow{\text{measure } A=i} |i\rangle_A \otimes \frac{1}{\sqrt{P(i)}} \sum_j c_{ij} |j\rangle_B$$

where $P(i) = \sum_j |c_{ij}|^2$.

3.6.2. Example: Measuring GHZ₃

Start with: $|\text{GHZ}_3\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$

Measure qubit 1 in Z-basis:

- **Outcome 0** (probability $\frac{1}{2}$):

$$|\text{GHZ}_3\rangle \rightarrow |0\rangle_1 \otimes |00\rangle_{23}$$

Qubits 2 and 3 are now in a **product state** $|00\rangle$ (no entanglement!)

- **Outcome 1** (probability $\frac{1}{2}$):

$$|\text{GHZ}_3\rangle \rightarrow |1\rangle_1 \otimes |11\rangle_{23}$$

Again, qubits 2 and 3 are in a product state $|11\rangle$.

Key observation: GHZ entanglement is completely destroyed by measuring a single qubit!

3.6.3. Example: Measuring W_3

Start with: $|W_3\rangle = \frac{1}{\sqrt{3}}(|100\rangle + |010\rangle + |001\rangle)$

Measure qubit 1 in **Z**-basis:

- **Outcome 0** (probability $\frac{2}{3}$):

$$|W_3\rangle \rightarrow |0\rangle_1 \otimes \frac{1}{\sqrt{2}}(|10\rangle + |01\rangle)_{23} = |0\rangle_1 \otimes |\Psi^+\rangle_{23}$$

Qubits 2 and 3 are in a **Bell state** (still entangled!)

- **Outcome 1** (probability $\frac{1}{3}$):

$$|W_3\rangle \rightarrow |1\rangle_1 \otimes |00\rangle_{23}$$

Product state (no entanglement).

Key observation: W entanglement survives the loss of one qubit (with probability $\frac{2}{3}$).

3.7. Tensor Network Visualization

For large n -qubit states, we can use **tensor network diagrams** to visualize entanglement structure.

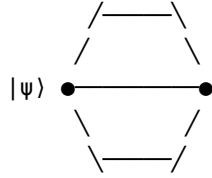
3.7.1. Product State

$$|0\rangle \text{ --- } |0\rangle \text{ --- } |0\rangle \text{ --- } |0\rangle$$

No connections $\rightarrow$ no entanglement.

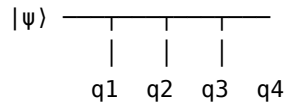
3. *n*-Qubit Systems

3.7.2. GHZ State



Central node connected to all qubits $\rightarrow$ maximal multipartite entanglement.

3.7.3. W State



Distributed structure $\rightarrow$ robust, partial entanglement.

(Note: These are simplified diagrams. Full tensor networks are more complex.)

3.8. Computational Complexity

3.8.1. State Vector Simulation

Classical simulation of n -qubit states requires:

- **Memory:** $O(2^n)$ complex numbers
- **Operations:** $O(2^n)$ per gate

Limitations:

3.9. Creating n -Qubit States in QRL

- **~45 qubits:** Maximum on world's best supercomputers (~ 35 TB RAM)
- **50+ qubits:** Classically intractable to simulate in general

This is where quantum computers achieve advantage!

3.8.2. Tensor Network Methods

For certain states (e.g., matrix product states), we can simulate efficiently using **tensor networks**:

- **Memory:** $O(n\chi^2)$ where χ is bond dimension
- **Works for:** 1D systems, weakly entangled states, ground states of local Hamiltonians

Limitation: Highly entangled states (like GHZ, W) require exponential resources even for tensor networks.

3.9. Creating n -Qubit States in QRL

In the Quantum Relational Language, you can create GHZ and W states directly:

```
from qrl import QRLProgram

program = QRLProgram("n-Qubit Entanglement")

# Create 4 qubits
q0, q1, q2, q3 = [program.create_system() for _ in range(4)]

# GHZ state (default)
ghz4 = program.entangle(q0, q1, q2, q3)
```

3. *n*-Qubit Systems

```
print(f"GHZ entropy: {ghz4.entanglement_entropy:.3f}") # 1.000

# W state
w4 = program.entangle(q0, q1, q2, q3, state_type="w")
print(f"W entropy: {w4.entanglement_entropy:.3f}") # < 1.000
```

This **relations-first** approach treats entanglement as a primitive operation, not a derived concept!

3.10. Summary

In this chapter, you learned:

- ✓ *n*-qubit systems live in 2^n -dimensional Hilbert spaces
- ✓ GHZ states: maximal, fragile, all-or-nothing entanglement
- ✓ W states: robust, partial, distributed entanglement
- ✓ GHZ and W are LOCC-inequivalent (different entanglement classes)
- ✓ Genuine multipartite entanglement involves all bipartitions
- ✓ Partial measurements affect entanglement differently for GHZ vs W
- ✓ Classical simulation becomes intractable beyond ~45 qubits

Part I Complete! You now have the quantum foundations needed to understand MBQC and QRL.

3.11. Exercises

3.11.1. Exercise 3.1: GHZ State Verification

For $|\text{GHZ}_4\rangle = \frac{1}{\sqrt{2}}(|0000\rangle + |1111\rangle)$:

(a) Calculate the probability of measuring outcome $|0101\rangle$ in the Z-basis.

(b) If you measure qubits 1 and 2 and get outcomes 0 and 1 respectively, what is the state of qubits 3 and 4?

Solution →

3.11.2. Exercise 3.2: W State Properties

For $|W_4\rangle = \frac{1}{2}(|1000\rangle + |0100\rangle + |0010\rangle + |0001\rangle)$:

(a) Calculate $P(1000)$ when all qubits are measured in the Z-basis.

(b) After measuring qubit 1 and getting 0, prove that qubits 2, 3, 4 are in state $|W_3\rangle$.

Solution →

3.11.3. Exercise 3.3: Entanglement Entropy Calculation

Calculate the bipartite entanglement entropy for $|W_3\rangle$ with partition 12|3 (qubits 1-2 vs qubit 3).

Hint: First find the Schmidt decomposition for this bipartition.

Solution →

3.11.4. Exercise 3.4: LOCC Impossibility

Explain intuitively why you cannot convert $|\text{GHZ}_3\rangle$ to $|W_3\rangle$ using only local operations and classical communication.

Solution →

3. *n*-Qubit Systems

3.11.5. Exercise 3.5: Multipartite Correlation

Consider the state:

$$|\psi\rangle = \frac{1}{2}(|000\rangle + |011\rangle + |101\rangle + |110\rangle)$$

(a) Is this state genuinely tripartite entangled? Check all bipartitions.

(b) Calculate the entanglement entropy for partition 1|23.

Solution →

3.11.6. Exercise 3.6: Fragility vs Robustness

Design an experiment (in words) that demonstrates the difference in robustness between GHZ and W states when qubits are lost.

Solution →

3.11.7. Exercise 3.7: n-Qubit Generalization

Generalize the formula for $|W_n\rangle$ and prove that measuring k qubits (with all outcomes 0) leaves the remaining qubits in state $|W_{n-k}\rangle$ (properly normalized).

Solution →

Next: Part II: Measurement-Based Quantum Computing

Part I complete! You're now ready to explore MBQC and QRL.

Part II.

**Part II: Measurement-Based
Quantum Computing**

4. MBQC Foundations

Measurement-Based Quantum Computing (MBQC) represents a radical departure from the circuit model. Instead of applying gates to qubits, we prepare a highly entangled resource state and perform the computation entirely through **measurements**.

This is not just a theoretical curiosity—it’s the foundation of QRL’s design philosophy and the native model for photonic quantum computers.

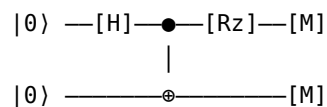
4.1. The Circuit Model vs MBQC

4.1.1. Traditional Circuit Model

In the gate-based (circuit) model:

1. **Initialize** qubits to $|0\rangle$
2. **Apply gates** sequentially: H , $CNOT$, $R_z(\theta)$, etc.
3. **Measure** at the end to get results

Representation:



This is how Qiskit, Cirq, and Q# work.

4. MBQC Foundations

4.1.2. Measurement-Based Quantum Computing

In MBQC:

1. **Prepare** a large entangled state (cluster state) upfront
2. **Measure** individual qubits in specific bases
3. **Adapt** subsequent measurement bases based on earlier results
4. **Read out** final qubits for the answer

Key insight: Computation happens through *measurement choices*, not gate applications!

4.2. The One-Way Quantum Computer

Proposed by Raussendorf and Briegel (2001), the one-way quantum computer operates in three steps:

One-Way QC Protocol

Step 1: Preparation

- Create a **cluster state** - a specific highly-entangled state on a lattice of qubits

Step 2: Measurement

- Measure qubits one-by-one in bases determined by:
 - The computation you want to perform
 - Outcomes of previous measurements (adaptive)

Step 3: Readout

- The remaining unmeasured qubits contain the result
- Apply classical post-processing (Pauli corrections) based on measurement record

4.3. Cluster States: The Universal Resource

Why “one-way”? The entangled resource state is consumed by measurement - you can’t reverse the process. Computation flows in one direction.

4.3. Cluster States: The Universal Resource

4.3.1. Definition

A **cluster state** is defined on a graph $G = (V, E)$ where:

- Each vertex represents a qubit
- Edges define entanglement

Preparation:

1. Initialize all qubits to $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$:

$$|\psi_0\rangle = |+\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$$

2. Apply CZ gate (controlled-Z) to each edge:

$$|\text{Cluster}_G\rangle = \prod_{(i,j) \in E} CZ_{ij} \cdot |\psi_0\rangle$$

where $CZ|a\rangle|b\rangle = (-1)^{ab}|a\rangle|b\rangle$.

4.3.2. Example: 3-Qubit Chain

For a linear graph: 1—2—3

Preparation:

$$|+\rangle^{\otimes 3} \xrightarrow{CZ_{12}} \xrightarrow{CZ_{23}} |\text{Cluster}\rangle$$

4. MBQC Foundations

Explicit state:

$$|\text{Cluster}\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle - |110\rangle + |111\rangle)$$

This looks complicated, but has a beautiful structure when viewed through stabilizers (Chapter 5).

4.3.3. Cluster State Properties

1. **Highly entangled:** Every qubit is entangled with its neighbors
2. **Translation invariant:** For infinite lattices, the state looks the same everywhere
3. **Universal:** Any quantum computation can be performed on a 2D cluster state
4. **Deterministic:** Creating cluster states is straightforward (unlike creating GHZ states, which require post-selection)

4.4. Measurement Patterns

A **measurement pattern** specifies how to perform computation on a cluster state.

4.4.1. Pattern Specification

A pattern consists of:

1. **Graph G** - defines the cluster state
2. **Measurement angles $\{\theta_i\}$** - basis for measuring each qubit
3. **Dependencies** - which measurements depend on previous outcomes

4. **Corrections** - Pauli corrections to apply based on measurement record

4.4.2. Measurement Bases

Qubits are typically measured in the **XY-plane** of the Bloch sphere:

$$|\pm_\theta\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm e^{i\theta}|1\rangle)$$

Special cases:

- $\theta = 0$: X-basis ($|+\rangle, |-\rangle$)
- $\theta = \frac{\pi}{2}$: Y-basis ($|+i\rangle, |-i\rangle$)
- $\theta = \pi$: $-X$ -basis

The choice of θ encodes the operation being performed.

4.4.3. Adaptive Measurements

Crucially, measurement angles can depend on previous outcomes:

$$\theta'_i = \theta_i + \pi \cdot f(s_1, s_2, \dots, s_{i-1})$$

where $s_j \in \{0, 1\}$ is the outcome of measuring qubit j , and f is a Boolean function.

This **feed-forward** is what allows MBQC to simulate any quantum circuit.

4. MBQC Foundations

4.5. Example: Single-Qubit Rotation

Let's see how MBQC implements a rotation gate $R_z(\theta)$ on a logical qubit.

4.5.1. Setup

Cluster: 3-qubit chain: 1—2—3

Goal: Apply $R_z(\theta)$ to the state encoded in qubit 1, output in qubit 3

4.5.2. Protocol

1. **Encode input:**

- Initialize cluster with input state on qubit 1

2. **Measure qubit 1:**

- Basis: X ($\theta = 0$)
- Outcome: $s_1 \in \{0, 1\}$

3. **Measure qubit 2:**

- Basis: $|\pm_\theta\rangle$ where θ is the rotation angle
- Adaptive angle: $\theta' = \theta + \pi s_1$ (depends on s_1 !)
- Outcome: $s_2 \in \{0, 1\}$

4. **Correct qubit 3:**

- Apply Pauli $X^{s_1}Z^{s_2}$ to qubit 3
- Qubit 3 now contains $R_z(\theta)|\psi_{\text{in}}\rangle$

Result: The rotation $R_z(\theta)$ has been performed entirely by measurements!

4.6. Universality of MBQC

Theorem: MBQC is Universal (Raussendorf & Briegel, 2001)

Any quantum circuit can be simulated on a 2D cluster state by choosing appropriate measurement angles and feed-forward rules.

More precisely: The one-way quantum computer is **computationally equivalent** to the circuit model.

4.6.1. Why This Matters

1. **Alternative computation model:** Not just a different implementation, but a fundamentally different way to think about quantum computing
2. **Hardware-native for photonics:** Photonic qubits are easier to create entangled than to gate directly - MBQC is their natural model
3. **Fault-tolerance:** Topological quantum computing (surface codes) is essentially MBQC
4. **QRL foundation:** Relations-first programming maps naturally to MBQC

4.7. Gate Teleportation

A key technique in MBQC is **gate teleportation** - implementing gates by teleporting through entangled resource states.

4. MBQC Foundations

4.7.1. Bell Measurement Teleportation

Recall quantum teleportation:

1. Share Bell pair $|\Phi^+\rangle_{23}$ between Alice and Bob
2. Alice performs Bell measurement on qubits 1-2
3. Bob applies Pauli corrections to qubit 3
4. Result: State of qubit 1 is now on qubit 3

4.7.2. Teleportation with Rotated Basis

If Alice measures in a **rotated** Bell basis, she effectively applies a gate during teleportation:

Modified protocol:

1. Bell pair $|\Phi^+\rangle_{23}$
2. Alice rotates qubit 2 by $R(\theta)$ before Bell measurement
3. Bob applies corrections
4. Result: $R(\theta)|\psi\rangle$ appears on qubit 3

This is the core principle behind MBQC: **measurements in rotated bases implement gates.**

4.8. Measurement Calculus

MBQC has a formal calculus for reasoning about measurement patterns.

4.8.1. Commands

Preparation:

```
N_i    # Create qubit i in state |+⟩
E_ij   # Entangle qubits i and j with CZ
```

Measurement:

```
M_i^θ  # Measure qubit i in basis  $|\pm_\theta\rangle$ 
```

Corrections:

```
X_i^s  # Apply Pauli X to i if signal s = 1
Z_i^s  # Apply Pauli Z to i if signal s = 1
```

4.8.2. Example Pattern

Single-qubit rotation $R_z(\theta)$:

```
N_1  N_2  N_3    # Prepare 3 qubits
E_12 E_23        # Create chain
M_1^θ → s_1      # Measure in X-basis
M_2^θ → s_2      # Measure at angle θ
X_3^s_1 Z_3^s_2  # Apply corrections
```

Output: Qubit 3 contains $R_z(\theta)|\psi\rangle$

4.9. Advantages of MBQC

4.9.1. For Photonic Quantum Computing

Why photonics loves MBQC:

1. **Easy entanglement generation:** Fusion gates create cluster states efficiently
2. **No photon-photon gates needed:** All operations are single-qubit measurements
3. **Loss tolerance:** Can teleport around lost photons
4. **Scalability:** Massively parallel cluster state generation

This is why companies like **PsiQuantum** and **Xanadu** are pursuing MBQC.

4.9.2. For Quantum Programming

Why QRL uses MBQC:

1. **Relations-first:** Entanglement is prepared upfront, not derived from gates
2. **Declarative:** Specify what you want, not how to implement it
3. **Composable:** Measurement patterns compose naturally
4. **Graph-based:** Visual intuition through cluster state graphs

4.10. MBQC vs Circuits: Complexity Comparison

Aspect	Circuit Model	MBQC
Initialization	n qubits in $ 0\rangle$	$m \gg n$ qubits in cluster state

4.11. Why MBQC Matters for Theory

Aspect	Circuit Model	MBQC
Computation	$O(\text{poly})$ gates	$O(\text{poly})$ measurements
Entanglement	Created by gates	Pre-existing in resource state
Space overhead	1x	~5x (ancilla qubits)
Time overhead	1x	1x (same circuit depth)
Classical processing	Minimal	Feed-forward required

Key tradeoff: MBQC uses more qubits but may be easier to implement in certain hardware.

4.11. Why MBQC Matters for Theory

MBQC reveals deep connections between:

1. Entanglement and Computation:

- Computational power is “stored” in entanglement
- Measurement “releases” this power

2. Information Flow:

- Quantum information flows through the cluster via measurement
- Classical information flows back via feed-forward

3. Quantum Contextuality:

- Measurement basis choice determines the computation
- Same state, different measurements = different algorithms

4.12. Summary

In this chapter, you learned:

✓ MBQC performs computation through measurement, not gates ✓ Cluster states are universal resource states for quantum computing ✓ Measurement patterns specify computations on cluster states ✓ Adaptive measurements enable universal quantum computation ✓ Gate teleportation is the key technique for implementing operations ✓ MBQC is the natural model for photonic quantum computers ✓ QRL is built on MBQC principles (relations-first)

4.13. Exercises

4.13.1. Exercise 4.1: Cluster State Preparation

For a 2-qubit cluster state on graph 1—2:

- (a) Write the initial state $|+\rangle^{\otimes 2}$ as a sum over computational basis states.
- (b) Apply CZ_{12} and write the resulting cluster state.
- (c) Verify this state is entangled by attempting to factor it.

Solution →

4.13.2. Exercise 4.2: Measurement Outcomes

For the 3-qubit chain cluster state, if you measure qubit 1 in the X-basis and get outcome 0, what is the state of qubits 2-3?

Solution →

4.13.3. Exercise 4.3: Gate Teleportation

Explain how measuring Alice's qubit in the Hadamard basis (instead of computational basis) during teleportation effectively applies a Hadamard gate to the teleported state.

Solution →

4.13.4. Exercise 4.4: Pattern Design

Design a measurement pattern to implement a Hadamard gate on a 3-qubit chain cluster state. Specify:

- Which qubit to measure first
- Measurement basis
- Any corrections needed

Solution →

4.13.5. Exercise 4.5: Resource Scaling

A circuit with depth d and width n requires approximately $d \cdot n$ qubits in MBQC (due to ancillas). For Grover's algorithm on n qubits with $O(\sqrt{2^n})$ gates, estimate the cluster state size needed.

Solution →

4.13.6. Exercise 4.6: Advantage Analysis

List three scenarios where MBQC might have a practical advantage over the circuit model, and three where the circuit model might be preferable.

Solution →

4. MBQC Foundations

Next: [Chapter 5: Cluster States](#) - Deep dive into graph states and stabilizer formalism

5. Cluster States

Cluster states are the workhorses of MBQC. In this chapter, we'll explore their mathematical structure using the **stabilizer formalism** - a powerful framework that makes cluster states computationally tractable despite their exponential complexity.

5.1. Graph States: General Framework

Cluster states are special cases of **graph states** - a family of entangled states defined by simple graphs.

5.1.1. Definition

Graph State Definition

Given an undirected graph $G = (V, E)$ with n vertices (qubits), the **graph state** $|G\rangle$ is:

$$|G\rangle = \prod_{(i,j) \in E} CZ_{ij} \cdot |+\rangle^{\otimes n}$$

where:

- $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ is the Hadamard state
- CZ_{ij} is the controlled-Z gate between qubits i and j
- The product is over all edges in the graph

5. Cluster States

Key properties:

- **Order-independent:** CZ_{ij} gates commute, so edge order doesn't matter
- **Self-inverse:** $CZ_{ij}^2 = I$ (applying CZ twice does nothing)
- **Local unitaries + entanglement:** Start with product state, entangle via graph structure

5.1.2. Examples

Single qubit (no edges):

$$|G\rangle = |+\rangle$$

Two qubits, one edge (1—2):

$$|G\rangle = CZ_{12}(|+\rangle \otimes |+\rangle) = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$$

Triangle (1—2—3—1):

$$|G\rangle = CZ_{12} \cdot CZ_{23} \cdot CZ_{13} \cdot |+\rangle^{\otimes 3}$$

5.2. Cluster States vs General Graph States

Cluster state specifically refers to graph states on **lattice graphs** (regular grids):

- **1D cluster:** Linear chain 1—2—3—...— n
- **2D cluster:** Square lattice (like a checkerboard)
- **3D cluster:** Cubic lattice

These regular structures have special properties useful for universal quantum computation.

5.3. The Stabilizer Formalism

Directly computing graph state amplitudes requires $O(2^n)$ operations. The **stabilizer formalism** provides a polynomial-time classical representation.

5.3.1. Pauli Operators

Recall the single-qubit Pauli operators:

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

Properties:

- $X^2 = Y^2 = Z^2 = I$
- $XY = iZ, \quad YZ = iX, \quad ZX = iY$
- $\{X, Z\} = 0$ (anticommute)

5.3.2. n-Qubit Pauli Group

For n qubits, the **Pauli group** $\mathcal{P}_n$ consists of all tensor products of single-qubit Paulis with phases $\pm 1, \pm i$:

$$\mathcal{P}_n = \{\pm 1, \pm i\}^\times \cdot \{I, X, Y, Z\}^{\otimes n}$$

Example for 2 qubits:

$$X \otimes Z, \quad I \otimes Y, \quad X \otimes X, \quad \dots$$

Shorthand notation: $XZ, IY, XX, \dots$ (omit $\otimes$)

5. Cluster States

5.3.3. Stabilizers

Stabilizer Definition

An operator S **stabilizes** a state $|\psi\rangle$ if:

$$S|\psi\rangle = |\psi\rangle$$

The set of all operators that stabilize $|\psi\rangle$ forms the **stabilizer group** of the state.

Example: Z stabilizes $|0\rangle$:

$$Z|0\rangle = |0\rangle \checkmark$$

but not $|1\rangle$:

$$Z|1\rangle = -|1\rangle \neq |1\rangle$$

5.3.4. Stabilizer States

A state is a **stabilizer state** if:

1. It has a stabilizer group generated by n independent Pauli operators
2. These generators commute with each other

Key theorem: An n -qubit stabilizer state is **uniquely determined** by n independent stabilizers.

This means we can represent the state by $O(n^2)$ bits instead of $O(2^n)$ complex amplitudes!

5.4. Graph State Stabilizers

For a graph state $|G\rangle$ on graph $G = (V, E)$, the stabilizers have a beautiful form.

Graph State Stabilizer Generators

For each vertex $i \in V$, define:

$$K_i = X_i \prod_{j \in N(i)} Z_j$$

where $N(i)$ is the set of neighbors of i in the graph.

These n operators generate the stabilizer group of $|G\rangle$, meaning:

$$K_i |G\rangle = |G\rangle \quad \text{for all } i$$

5.4.1. Example: 2-Qubit Chain

Graph: 1—2

Neighbors:

- $N(1) = \{2\}$
- $N(2) = \{1\}$

Stabilizers:

$$K_1 = X_1 Z_2$$

$$K_2 = Z_1 X_2$$

Verification: Let's prove $K_1 |G\rangle = |G\rangle$ where $|G\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$.

5. Cluster States

$$\begin{aligned} K_1|G\rangle &= X_1 Z_2 \cdot \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle) \\ &= \frac{1}{2} X_1(|00\rangle - |01\rangle + |10\rangle + |11\rangle) \\ &= \frac{1}{2}(|10\rangle - |11\rangle + |00\rangle + |01\rangle) \\ &= \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle) \\ &= |G\rangle \quad \checkmark \end{aligned}$$

5.4.2. Example: 3-Qubit Chain

Graph: 1—2—3

Neighbors:

- $N(1) = \{2\}$
- $N(2) = \{1, 3\}$
- $N(3) = \{2\}$

Stabilizers:

$$\begin{aligned} K_1 &= X_1 Z_2 \\ K_2 &= Z_1 X_2 Z_3 \\ K_3 &= Z_2 X_3 \end{aligned}$$

These three operators completely characterize the 3-qubit cluster state!

5.5. Why Stabilizers Are Powerful

5.5.1. Efficient Classical Representation

State vector approach:

5.5. *Why Stabilizers Are Powerful*

- Store 2^n complex amplitudes
- Requires exponential memory

Stabilizer approach:

- Store n stabilizer generators
- Each generator is n Pauli operators
- Total: $O(n^2)$ bits (polynomial!)

5.5.2. Example: 100-Qubit Cluster

State vector:

- $2^{100} \approx 10^{30}$ complex amplitudes
- 10^{31} bytes (more atoms than in the observable universe!)

Stabilizers:

- 100 stabilizer generators
- Each: 100 Pauli operators
- $100 \times 100 = 10,000$ symbols (trivial to store!)

5.5.3. Efficient Simulation

Operations on stabilizer states can be simulated efficiently:

- **Clifford gates:** Update stabilizers in $O(n^2)$ time
- **Pauli measurements:** Sample outcome and update stabilizers in $O(n^2)$ time
- **Clifford circuits:** Full simulation in $O(n^3)$ time per gate

5. Cluster States

Limitation: Non-Clifford gates (like T -gate, arbitrary rotations) take exponential time.

MBQC uses mostly Pauli measurements (efficient) plus adaptive basis choices (can be non-Clifford).

5.6. Local Complementation

Graph states have a fascinating property: local operations can transform one graph into another.

Local Complementation Rule

Applying a Hadamard gate to qubit i of graph state $|G\rangle$ produces a new graph state $|G'\rangle$ where G' is obtained by **complementing** the induced subgraph on $N(i)$ (neighbors of i).

Complement: If edge (j, k) exists, remove it; if it doesn't exist, add it.

5.6.1. Example

Graph G :

1 – 2 – 3

Apply H_2 (Hadamard on qubit 2):

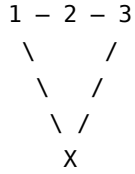
Neighbors of 2: $N(2) = \{1, 3\}$

Complement induced subgraph on $\{1, 3\}$:

- Currently no edge $(1, 3) \rightarrow$ add edge $(1, 3)$

5.7. Creating Cluster States in Practice

Resulting graph G' :



(Triangle: $1-2-3-1$)

State transformation:

$$H_2|G\rangle = |G'\rangle$$

This allows us to **transform between graph states using only local operations!**

5.7. Creating Cluster States in Practice

5.7.1. Deterministic Preparation

Circuit method (what we've defined):

1. Initialize $|+\rangle^{\otimes n}$
2. Apply CZ gates according to graph G

Problem: CZ gates are hard to implement in some platforms (especially photonics).

5. Cluster States

5.7.2. Photonic Preparation: Fusion Gates

For photonic qubits, cluster states are created using **fusion gates**:

1. **Generate Bell pairs:** Create many $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ states
2. **Fuse pairs:** Perform Bell measurements between qubits from different pairs
3. **Success:** With probability $\frac{1}{2}$, fusion creates a larger entangled state
4. **Repeat:** Fuse many pairs to create large cluster states

Advantage: Only requires single-photon sources and linear optical elements (beam splitters, detectors).

This is why photonic quantum computers use MBQC!

5.8. Measuring Graph States

5.8.1. Single-Qubit Measurement

When you measure qubit i in a graph state:

Z-basis measurement (outcome $s \in \{0, 1\}$):

- Remove qubit i from graph
- Apply Z^s correction to all neighbors of i

X-basis measurement:

- Remove qubit i and all edges touching i
- No corrections needed
- Remaining qubits: still in a graph state!

Y-basis measurement:

5.8. Measuring Graph States

- More complex update rules
- Used less frequently in practice

5.8.2. Example: Measuring 3-Qubit Chain

Graph: 1—2—3

Measure qubit 1 in X-basis:

- Remove qubit 1
- Remove edge 1—2
- **Remaining graph:** 2—3 (2-qubit chain)
- **No corrections**

The state is now a 2-qubit cluster state on qubits 2-3!

5.8.3. Adaptive Measurements

In MBQC, measurement bases adapt based on previous outcomes:

Example pattern:

1. Measure qubit 1 in X $\rightarrow$ outcome s_1
2. Measure qubit 2 in basis $|\pm_{\theta}\rangle$ where $\theta' = \theta + \pi s_1$
3. Apply corrections based on s_1, s_2

This feed-forward is what makes MBQC universal.

5. Cluster States

5.9. Stabilizers Under Measurement

When measuring qubit i , stabilizers update:

Before measurement: Stabilizers $\{K_1, \dots, K_n\}$

After measuring qubit i with outcome s :

1. **Fix outcome:** Replace K_i with $(-1)^s Z_i$ (Z-basis) or $(-1)^s X_i$ (X-basis)
2. **Propagate:** Update other stabilizers by multiplying with K_i to eliminate dependence on qubit i
3. **Reduce:** New stabilizer group has $n - 1$ generators (one qubit measured out)

This keeps the stabilizer representation efficient throughout the measurement process.

5.10. 2D Cluster States and Universality

Universal Resource Theorem

Any quantum circuit of depth d on n qubits can be simulated on a **2D cluster state** of size $O(d \times n)$ using only single-qubit measurements.

Why 2D specifically?

- 1D clusters can only simulate constant-depth circuits
- 2D clusters are universal for any depth
- 3D clusters offer no additional computational power

Practical significance: A large 2D cluster state is a “universal quantum computer in a box” - all computation is performed by measurement choices.

5.11. Summary

In this chapter, you learned:

✓ Graph states are defined by simple graphs and CZ gates ✓ Stabilizer formalism provides polynomial classical representation ✓ Graph state stabilizers: $K_i = X_i \prod_{j \in N(i)} Z_j$ ✓ Stabilizers enable efficient simulation of Clifford operations ✓ Local complementation transforms graphs via local unitaries ✓ Photonic cluster states created via fusion gates ✓ Measuring graph states preserves graph structure ✓ 2D cluster states are universal for quantum computation

5.12. Exercises

5.12.1. Exercise 5.1: Stabilizer Verification

For the 2-qubit graph state on 1—2, verify that $K_2 = Z_1 X_2$ stabilizes:

$$|G\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$$

Solution →

5.12.2. Exercise 5.2: 4-Qubit Chain Stabilizers

Write down the four stabilizer generators for a 4-qubit linear cluster state:

$$1-2-3-4$$

Solution →

5. Cluster States

5.12.3. Exercise 5.3: Stabilizer Commutation

Prove that the stabilizers $K_1 = X_1 Z_2$ and $K_2 = Z_1 X_2 Z_3$ commute:

$$K_1 K_2 = K_2 K_1$$

Solution →

5.12.4. Exercise 5.4: Local Complementation

Starting with a 4-qubit path graph 1—2—3—4, apply Hadamard to qubit 3 using local complementation rules. Draw the resulting graph.

Solution →

5.12.5. Exercise 5.5: Measurement Update

For a 3-qubit chain 1—2—3 with stabilizers:

$$K_1 = X_1 Z_2, \quad K_2 = Z_1 X_2 Z_3, \quad K_3 = Z_2 X_3$$

If qubit 1 is measured in the X-basis with outcome 0, what are the stabilizers for the remaining 2-qubit state?

Solution →

5.12.6. Exercise 5.6: Graph State Entanglement

For a star graph (one central qubit connected to n outer qubits), calculate the entanglement entropy between the central qubit and the rest.

Solution →

5.12.7. Exercise 5.7: Resource Comparison

Estimate the number of qubits needed in a 2D cluster state to simulate:
(a) Grover's algorithm on 10 qubits (b) Quantum Fourier Transform on 20 qubits

Solution →

Next: [Chapter 6: Measurement Patterns](#) - Compiling circuits to MBQC

6. Measurement Patterns

The power of MBQC lies in its ability to simulate any quantum circuit using only measurements. In this chapter, we'll see exactly how to **compile** quantum circuits into measurement patterns—the bridge from gate-based algorithms to measurement-based execution.

6.1. From Circuits to Patterns

6.1.1. The Compilation Problem

Given: A quantum circuit with gates $U_1, U_2, \dots, U_d$ acting on n qubits

Goal: Find a measurement pattern that:

1. Prepares a cluster state
2. Measures qubits in specific bases
3. Applies corrections
4. Produces the same output as the circuit

Key challenge: Gates are deterministic, but measurements are probabilistic. How do we ensure the correct output?

Answer: Adaptive measurements and Pauli corrections.

6.2. Gate Teleportation: The Core Technique

Every gate in MBQC is implemented via **gate teleportation** - a generalization of quantum teleportation where the gate is “encoded” in the measurement basis.

6.2.1. Standard Teleportation Recap

Setup: Alice and Bob share $|\Phi^+\rangle_{23} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$

Protocol:

1. Alice has input $|\psi\rangle_1$
2. Alice performs Bell measurement on qubits 1-2 $\rightarrow$ outcomes (s_x, s_z)
3. Bob applies $X^{s_x}Z^{s_z}$ to qubit 3
4. Qubit 3 now contains $|\psi\rangle$

Key insight: The state is transferred, but **no gate** is applied.

6.2.2. Teleporting Through a Gate

Modified setup: Alice and Bob share a modified entangled state:

$$|\text{resource}\rangle = (I \otimes U)|\Phi^+\rangle$$

where U is the gate we want to apply.

Protocol:

1. Alice performs Bell measurement $\rightarrow$ outcomes (s_x, s_z)
2. Bob applies corrections $X^{s_x}Z^{s_z}$
3. Result: $U|\psi\rangle$ appears on Bob’s qubit!

6.3. Compilation Rules for Single-Qubit Gates

Proof: Gate Teleportation

Claim: Teleporting through $(I \otimes U)|\Phi^+\rangle$ applies gate U .

Proof sketch:

Initial state:

$$|\psi\rangle_1 \otimes (I \otimes U)|\Phi^+\rangle_{23}$$

After Bell measurement (outcome $|s_x s_z\rangle$):

$$|s_x s_z\rangle_{12} \otimes X^{s_x} Z^{s_z} U|\psi\rangle_3$$

After corrections:

$$|s_x s_z\rangle_{12} \otimes U|\psi\rangle_3 \quad \checkmark$$

The gate U has been applied! $\square$

6.3. Compilation Rules for Single-Qubit Gates

6.3.1. Rotations via Measurement Angles

For rotation gates $R_\alpha(\theta) = e^{-i\theta\alpha/2}$ where $\alpha \in \{X, Y, Z\}$:

MBQC implementation:

1. Add ancilla qubit in state $|+\rangle$
2. Entangle with CZ
3. Measure ancilla in basis $|\pm_\theta\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm e^{i\theta}|1\rangle)$
4. Apply Pauli correction based on outcome

Measurement basis determines the rotation angle!

6.3.2. Pauli Gates

X gate:

6. Measurement Patterns

- Measure in Z-basis ($\theta = \pi$)
- Always apply X correction (deterministic)

Z gate:

- Measure in X-basis ($\theta = 0$)
- Always apply Z correction (deterministic)

Y gate:

- Measure in basis at $\theta = \frac{\pi}{2}$
- Apply both X and Z corrections

6.3.3. Hadamard Gate

Circuit: H

MBQC pattern:

1. Measure in X-basis ($\theta = 0$)
2. If outcome is 1, apply Z correction

Result: Hadamard gate implemented!

6.3.4. Phase Gate ($S = R_Z(\pi/2)$)

Circuit: $S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$

MBQC pattern:

1. Measure in basis $|\pm_{\pi/4}\rangle$
2. Apply Z correction based on outcome

6.3.5. General Single-Qubit Unitary

Any single-qubit gate can be decomposed as:

$$U = e^{i\alpha} R_Z(\beta) R_Y(\gamma) R_Z(\delta)$$

MBQC implementation:

- 3 ancilla qubits (one per rotation)
- Measure each at appropriate angle
- Chain corrections through the pattern

6.4. Compilation Rules for Two-Qubit Gates

6.4.1. CNOT Gate

The CNOT gate is the workhorse of quantum circuits.

Circuit: $CNOT_{c,t}$ (control c , target t)

MBQC implementation:

Pattern:

Cluster: $c' - a - t'$
 $\begin{array}{cc} | & | \\ c & t \end{array}$

where:

- c, t are input qubits (control, target)
- c', t' are output qubits
- a is ancilla

6. Measurement Patterns

Measurements:

1. Measure c in X-basis $\rightarrow$ outcome s_c
2. Measure a in X-basis $\rightarrow$ outcome s_a
3. Measure t in X-basis $\rightarrow$ outcome s_t

Corrections:

- Apply Z^{s_c} to c'
- Apply X^{s_a} to t'
- Apply Z^{s_t} to t'

Result: $CNOT_{c',t'}$ applied to the initial state.

6.4.2. CZ Gate (Controlled-Z)

Simpler in MBQC! Since cluster states are built from CZ gates, this is native.

Pattern:

```

c' - t'
|   |
c   t

```

Measurements:

1. Measure c in X-basis $\rightarrow$ outcome s_c
2. Measure t in X-basis $\rightarrow$ outcome s_t

Corrections:

- Apply Z^{s_c} to t'
- Apply Z^{s_t} to c'

Note: CZ is symmetric (no control/target distinction).

6.5. Full Circuit Compilation

6.5.1. Step-by-Step Process

Input: Quantum circuit on n qubits with d layers of gates

Step 1: Circuit Decomposition

- Decompose all gates into: $\{H, R_Z(\theta), CNOT\}$ (universal gate set)
- Organize into layers (gates that can be applied in parallel)

Step 2: Graph Construction

- Create linear cluster for each qubit: n chains of length $d + 1$
- Add connections between chains for CNOT gates
- Result: 2D cluster state of size $\sim n \times d$

Step 3: Assign Measurement Angles

- For each gate, assign measurement basis to corresponding ancilla
- Single-qubit rotations $\rightarrow$ measurement angle = rotation angle
- Hadamard $\rightarrow$ X-basis measurement
- CNOT $\rightarrow$ X-basis measurements with appropriate corrections

Step 4: Determine Dependencies

- Identify feed-forward: which measurements depend on previous outcomes
- Chain corrections: later measurements depend on earlier outcomes

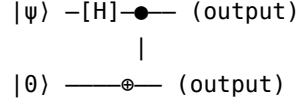
Step 5: Generate Pattern

- Output: List of measurements with angles and dependencies
- Corrections specified as functions of previous outcomes

6. Measurement Patterns

6.5.2. Example: Simple Circuit

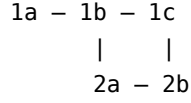
Circuit:



Step 1: Decomposition

- Already in $\{H, CNOT\}$

Step 2: Cluster Graph



where:

- 1a: input qubit 1 ($|\psi\rangle$)
- 2a: input qubit 2 ($|0\rangle$)
- 1b: ancilla for H
- 1c, 2b: ancillas for $CNOT$

Step 3: Measurements

- Measure 1a in X-basis (for H) $\rightarrow s_1$
- Measure 1b in X-basis $\rightarrow s_2$ (depends on s_1)
- Measure 2a in X-basis $\rightarrow s_3$

Step 4: Corrections

- On 1c: apply $Z^{s_1} X^{s_2}$
- On 2b: apply Z^{s_3}

Output: Qubits 1c and 2b contain the circuit output.

6.6. Adaptive Measurement

The power of MBQC comes from **adapting** measurement bases based on previous outcomes.

6.6.1. Feed-Forward Rules

For measurement i with nominal angle θ_i , the actual measurement basis is:

$$\theta'_i = \theta_i + \pi \cdot f_i(s_1, \dots, s_{i-1})$$

where f_i is a Boolean function (returns 0 or 1) of previous outcomes.

Example:

$$\theta'_3 = \theta_3 + \pi(s_1 \oplus s_2)$$

If $s_1 \oplus s_2 = 1$ (odd parity), flip the measurement angle by π .

6.6.2. Why This Works

Adaptive bases compensate for the randomness of earlier measurements:

- Measurement outcomes are random ($s_i \in \{0, 1\}$ with equal probability)
- But correlations between outcomes are deterministic
- Adapt later bases to maintain correct overall computation

This is analogous to error correction: adapt to “errors” (random outcomes) as you go.

6. Measurement Patterns

6.7. Pauli Corrections

After all measurements, apply **Pauli corrections** to output qubits based on the full measurement record.

6.7.1. Correction Functions

For each output qubit j , the correction is:

$$\sigma_j = X^{f_j^X(s_1, \dots, s_m)} Z^{f_j^Z(s_1, \dots, s_m)}$$

where f_j^X, f_j^Z are Boolean functions of all measurement outcomes.

Example:

$$\sigma_2 = X^{s_1 \oplus s_3} Z^{s_2}$$

6.7.2. Classical Post-Processing

These corrections can be applied:

1. **Physically:** Apply actual X and Z gates at the end
2. **Classically:** Track corrections and flip interpretation of final measurement outcomes

Option 2 is often preferred: No need for physical gates, just classical logic.

6.8. Optimization Techniques

6.8.1. Minimizing Cluster Size

Goal: Use fewest qubits possible

Techniques:

1. **Gate cancellation:** Simplify circuit before compilation
2. **Parallel gates:** Merge gates acting on different qubits
3. **Native gates:** Use CZ instead of $CNOT$ when possible (saves ancillas)

Example: A depth- d circuit on n qubits naively requires $n \times (d+1)$ qubits, but can often be reduced to $\sim n \times d/2$ with optimization.

6.8.2. Minimizing Measurement Depth

Measurement depth: Maximum number of measurements that must be performed sequentially (feed-forward dependencies).

Goal: Minimize measurement depth to reduce latency

Techniques:

1. **Parallelize independent measurements**
2. **Reorder circuit layers** to reduce dependencies
3. **Trade space for time:** Use more qubits to enable parallel measurements

6. Measurement Patterns

6.8.3. Deterministic Patterns

Some gates can be implemented **deterministically** (no corrections needed):

- **Pauli gates:** Always require specific corrections, but outcome is known
- $\pi/2$ **rotations:** Can be made deterministic with clever basis choices

Maximizing deterministic gates reduces classical processing overhead.

6.9. From Patterns to QRL

In the Quantum Relational Language, these patterns are **automatically generated** from high-level relational specifications.

6.9.1. QRL Abstraction Levels

Level 1: Relations (user writes this)

```
ghz = program.entangle(q0, q1, q2) # GHZ relation
```

Level 2: Graph Extraction (QRL infers)

Graph: $q0 - \text{ancilla1} - \text{ancilla2}$
 | |
 q1 q2

Level 3: Measurement Pattern (QRL compiles)

6.10. Measurement Pattern Language

```
Measure ancilla1 at  $\theta=0 \rightarrow s1$   
Measure ancilla2 at  $\theta=\pi/4 \rightarrow s2$   
Correct  $q0$  with  $X^{s1} Z^{s2}$ 
```

Level 4: Photonic Backend (QRL targets)

Strawberry Fields operations:

- Fusion gates to create cluster
- Homodyne measurements at angles
- Classical feed-forward

This **layered compilation** is the essence of QRL's design.

6.10. Measurement Pattern Language

Formally, patterns can be specified in a domain-specific language:

6.10.1. Commands

Entanglement:

```
N_i      # New qubit i in state  $|+\rangle$   
E_ij     # Entangle qubits i and j with CZ
```

Measurement:

```
M_i $^\alpha$     # Measure qubit i in basis  $|\pm_\alpha\rangle$   
[M_i $^\alpha$ ] $^s$   # Adaptive measurement ( $\alpha$  depends on signal s)
```

Corrections:

6. Measurement Patterns

```
X_i^s      # Apply X to i if signal s = 1
Z_i^s      # Apply Z to i if signal s = 1
```

6.10.2. Example Pattern

Hadamard gate:

```
N_1 N_2      # Two qubits
E_12         # Entangle them
M_1^0 → s    # Measure qubit 1 in X-basis
Z_2^s        # Correct qubit 2
```

Output: Qubit 2 contains $H|\psi\rangle$ where $|\psi\rangle$ was the input on qubit 1.

6.11. Equivalence to Circuit Model

Computational Equivalence Theorem

The MBQC model with measurement patterns is **computationally equivalent** to the quantum circuit model:

1. Any quantum circuit can be compiled to a measurement pattern
2. Any measurement pattern can be converted to a circuit
3. Both models have the same computational power (BQP)

Implication: MBQC is not just an implementation strategy - it's a fundamentally equivalent model of quantum computation.

6.12. Summary

In this chapter, you learned:

✓ Gate teleportation implements gates via measurement basis choices ✓
 Single-qubit rotations compiled to measurement angles ✓ CNOT compiled
 using 2D cluster structure ✓ Full circuits compiled to 2D cluster states of
 size $O(n \times d)$ ✓ Adaptive measurements enable deterministic computation
 despite random outcomes ✓ Pauli corrections fix up the final result based on
 measurement record ✓ Optimization reduces cluster size and measurement
 depth ✓ MBQC and circuits are computationally equivalent

Part II complete! You now understand the MBQC foundations that
 power QRL.

6.13. Exercises

6.13.1. Exercise 6.1: Hadamard Compilation

Write out the complete measurement pattern for implementing a Hadamard
 gate, including:

- Cluster preparation commands
- Measurement command with basis
- Correction commands

Solution →

6.13.2. Exercise 6.2: Rotation Angle

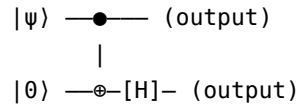
A rotation $R_Z(\theta)$ is implemented by measuring in basis $|\pm_\alpha\rangle$. What is the
 relationship between θ and α ?

6. Measurement Patterns

Solution →

6.13.3. Exercise 6.3: CNOT Pattern

Draw the cluster graph for implementing $CNOT$ followed by Hadamard on the target qubit:



Specify all measurement bases.

Solution →

6.13.4. Exercise 6.4: Adaptive Measurement

For a circuit with gates H , then $R_Z(\theta)$, if the first measurement (for H) gives outcome $s_1 = 1$, should the second measurement angle be θ or $\theta + \pi$? Justify your answer.

Solution →

6.13.5. Exercise 6.5: Pattern Optimization

A depth-10 circuit on 5 qubits would naively compile to a $5 \times 11 = 55$ qubit cluster. Describe two techniques to reduce this cluster size.

Solution →

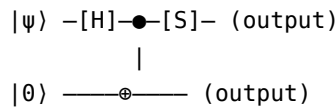
6.13.6. Exercise 6.6: Deterministic Gates

Prove that the Pauli X gate can be implemented deterministically in MBQC (meaning the correction is always the same, independent of measurement outcome).

Solution →

6.13.7. Exercise 6.7: Circuit to Pattern

Compile this circuit to a measurement pattern:



where $S = R_Z(\pi/2)$ is the phase gate.

Solution →

Congratulations! You've completed Parts I and II - the full theoretical foundation for quantum computing and MBQC.

Next steps:

- Revisit **Part III: QRL Programming** with your new theoretical understanding
- Explore **Part IV: Advanced Topics** (Chapters 12-15) when ready

You're now equipped to understand QRL's relations-first, MBQC-native approach to quantum programming!

Part III.

Part III: QRL Programming

7. Your First Bell State

Welcome to your first hands-on QRL tutorial! In this chapter, you'll create your first quantum program: a Bell state - one of the most fundamental entangled quantum states.

7.1. What is a Bell State?

A **Bell state** is a maximally entangled 2-qubit quantum state. The most common Bell state is:

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

This state has fascinating properties:

- **Superposition:** Equal probability of measuring $|00\rangle$ or $|11\rangle$
- **Perfect correlation:** If you measure qubit A as 0, qubit B will always be 0
- **Maximal entanglement:** Entropy $S = 1.0$ (maximum for 2 qubits)

7.2. Setting Up

First, make sure QRL is installed:

7. Your First Bell State

```
pip install quantum-relational-language
```

Create a new Python file `my_first_bell_state.py`:

```
from qrl import QRLProgram, create_question, QuestionType

# Create a quantum program
program = QRLProgram("My First Bell State")
```

7.3. Creating Quantum Systems

In QRL, you don't create individual "qubits" - you create **quantum systems** that can then be entangled into **relations**:

```
# Create two quantum systems
qubit_a = program.create_system()
qubit_b = program.create_system()

print(f"Created systems: {qubit_a.system_id} and {qubit_b.system_id}")
```

Each system gets a unique ID. At this point, they're independent - not yet entangled.

7.4. Entangling Systems

Here's where QRL's philosophy shines. Instead of applying gates like **CNOT**, you directly express the **relationship** you want:

7.5. Understanding the State

```
# Create a Bell pair (maximally entangled state)
bell_pair = program.entangle(qubit_a, qubit_b)

print(f"Bell state created!")
print(f"State vector shape: {bell_pair.state.shape}")
print(f"Entanglement entropy: {bell_pair.entanglement_entropy:.3f}")
```

Output:

```
Bell state created!
State vector shape: (4,)
Entanglement entropy: 1.000
```

The entropy of 1.0 confirms this is **maximally entangled**.

7.5. Understanding the State

Let's look at the actual quantum state:

```
import numpy as np

print("Bell state vector:")
print(bell_pair.state)
print()
print("In basis notation:")
print(f"|00⟩: {bell_pair.state[0]:.3f}")
print(f"|01⟩: {bell_pair.state[1]:.3f}")
print(f"|10⟩: {bell_pair.state[2]:.3f}")
print(f"|11⟩: {bell_pair.state[3]:.3f}")
```

Output:

7. Your First Bell State

Bell state vector:

```
[0.707+0j  0.+0j  0.+0j  0.707+0j]
```

In basis notation:

```
|00>: 0.707
```

```
|01>: 0.000
```

```
|10>: 0.000
```

```
|11>: 0.707
```

This is exactly $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$! (Note: $0.707 \approx \frac{1}{\sqrt{2}}$)

7.6. Complete Example

Here's the complete program you can run:

```
from qrl import QRLProgram, create_question, QuestionType

def main():
    # Create program
    program = QRLProgram("Complete Bell State Example")

    # Create quantum systems
    qubit_a = program.create_system()
    qubit_b = program.create_system()

    # Entangle into Bell state
    bell_pair = program.entangle(qubit_a, qubit_b)

    # Show state information
    print("=" * 50)
    print("BELL STATE CREATED")
    print("=" * 50)
```

7.7. What You Learned

```
print(f"State vector: {bell_pair.state}")
print(f"Entanglement entropy: {bell_pair.entanglement_entropy:.3f}")
print(f"Systems: {list(bell_pair.systems)}")

if __name__ == "__main__":
    main()
```

7.7. What You Learned

Congratulations! You've just:

✓ Created your first quantum program in QRL ✓ Generated a maximally entangled Bell state ✓ Understood entanglement entropy ✓ Explored the quantum state vector

Next: [Chapter 8: Understanding GHZ States](#) - Scale to 3+ qubits

8. Understanding GHZ States

In the previous chapter, you created your first Bell state - a 2-qubit maximally entangled state. Now we'll scale up to **3+ qubits** with GHZ states, one of the most important classes of multipartite entanglement.

8.1. What is a GHZ State?

GHZ (Greenberger-Horne-Zeilinger) states are maximally entangled states of 3 or more qubits. The 3-qubit GHZ state is:

$$|GHZ_3\rangle = \frac{|000\rangle + |111\rangle}{\sqrt{2}}$$

For n qubits:

$$|GHZ_n\rangle = \frac{|0\rangle^{\otimes n} + |1\rangle^{\otimes n}}{\sqrt{2}} = \frac{|00\dots 0\rangle + |11\dots 1\rangle}{\sqrt{2}}$$

8.2. Why GHZ States Matter

GHZ states are fundamental to quantum computing and quantum information:

1. **Quantum sensing** - Heisenberg-limited precision measurements

8. Understanding GHZ States

2. **Quantum communication** - Secret sharing protocols
3. **Bell's theorem** - Demonstrate quantum non-locality
4. **Quantum error correction** - Building blocks for logical qubits
5. **MBQC** - Resource states for measurement-based computation

8.3. GHZ vs Bell States

Property	Bell (2 qubits)	GHZ (3+ qubits)
State	$\frac{ 00\rangle + 11\rangle}{\sqrt{2}}$	$\frac{ 000\rangle + 111\rangle}{\sqrt{2}}$
Entanglement	Bipartite (2-way)	Multipartite (n-way)
Correlation	Perfect pairwise	Perfect n-way
Entropy	$S = 1.0$ (max)	$S = 1.0$ (max for 1 vs n-1 split)
Robustness	Loss of 1 qubit = no entanglement	Loss of 1 qubit = no entanglement

Key difference: GHZ states exhibit **genuine multipartite entanglement** - you can't decompose them into Bell pairs!

8.4. Creating a 3-Qubit GHZ State in QRL

In QRL, creating GHZ states is as simple as Bell states - just entangle more systems:

```
from qrl import QRLProgram

# Create program
program = QRLProgram("GHZ State Tutorial")
```

8.5. Examining the Quantum State

```
# Create 3 quantum systems
q0 = program.create_system()
q1 = program.create_system()
q2 = program.create_system()

# Entangle them into GHZ state
ghz3 = program.entangle(q0, q1, q2)

print("GHZ state created!")
print(f"State vector shape: {ghz3.state.shape}")
print(f"Number of qubits: {len(ghz3.systems)}")
print(f"Entanglement entropy: {ghz3.entanglement_entropy:.3f}")
```

Output:

```
GHZ state created!
State vector shape: (8,)
Number of qubits: 3
Entanglement entropy: 1.000
```

Notice:

- State vector has $2^3 = 8$ components (8-dimensional Hilbert space)
- Entropy is still 1.0 (maximal for 1-vs-2 split)

8.5. Examining the Quantum State

Let's look at the amplitudes:

8. Understanding GHZ States

```
import numpy as np

print("GHZ state amplitudes:")
print(ghz3.state)
print()

# Print in computational basis
basis_labels = ['|000>', '|001>', '|010>', '|011>',
                '|100>', '|101>', '|110>', '|111>']

for i, label in enumerate(basis_labels):
    amp = ghz3.state[i]
    prob = abs(amp)**2
    print(f"{label}: amplitude = {amp:.3f}, probability = {prob:.3f}")
```

Output:

```
GHZ state amplitudes:
[0.707+0j 0.+0j 0.+0j 0.+0j 0.+0j 0.+0j 0.+0j 0.707+0j]

|000>: amplitude = 0.707, probability = 0.500
|001>: amplitude = 0.000, probability = 0.000
|010>: amplitude = 0.000, probability = 0.000
|011>: amplitude = 0.000, probability = 0.000
|100>: amplitude = 0.000, probability = 0.000
|101>: amplitude = 0.000, probability = 0.000
|110>: amplitude = 0.000, probability = 0.000
|111>: amplitude = 0.707, probability = 0.500
```

Perfect! Only $|000\rangle$ and $|111\rangle$ have non-zero amplitudes, each with 50% probability.

8.6. GHZ Correlations: The Triple Coincidence

The magic of GHZ states is **perfect 3-way correlation**:

- If you measure all 3 qubits, they **always** give the same result
- Either all measure 0, or all measure 1
- Never mixed like 001 or 110

Let's verify:

```
from qrl import create_question, QuestionType

def test_ghz_correlations(num_trials=100):
    """Verify GHZ state 3-way correlations."""

    all_same = 0 # Count trials where all 3 qubits match

    for trial in range(num_trials):
        # Create fresh GHZ state
        program = QRLProgram(f"GHZ Trial {trial}")
        q0 = program.create_system()
        q1 = program.create_system()
        q2 = program.create_system()
        ghz = program.entangle(q0, q1, q2)

        # Measure all 3 qubits in Z basis
        question = create_question(QuestionType.SPIN_Z)
        alice = program.add_perspective("alice")
        result = program.ask(ghz, question, perspective="alice")

        # Check if all outcomes are the same
        outcomes = list(result.values())
        if outcomes[0] == outcomes[1] == outcomes[2]:
            all_same += 1
```

8. Understanding GHZ States

```
correlation = all_same / num_trials
print(f"Trials: {num_trials}")
print(f"All 3 qubits matched: {all_same}/{num_trials}")
print(f"Perfect correlation rate: {correlation:.1%}")
print()
print("Expected: 100% for GHZ state")

# Run the test
test_ghz_correlations(100)
```

Output:

```
Trials: 100
All 3 qubits matched: 100/100
Perfect correlation rate: 100.0%

Expected: 100% for GHZ state
```

Perfect 3-way correlation! This is genuine multipartite entanglement.

8.7. Scaling to 4+ Qubits

QRL supports arbitrary n-qubit GHZ states. Let's create a 5-qubit GHZ:

```
# Create 5-qubit GHZ state
program = QRLProgram("5-Qubit GHZ")

# Create 5 systems
qubits = [program.create_system() for _ in range(5)]
```

8.8. Partial Measurements on GHZ States

```
# Entangle all 5 into GHZ state
ghz5 = program.entangle(*qubits)

print(f"5-qubit GHZ created!")
print(f"State vector size: {ghz5.state.shape} (2^5 = 32 dimensions)")
print(f"Systems: {len(ghz5.systems)}")
print(f"Entanglement entropy: {ghz5.entanglement_entropy:.3f}")

# Check the state
print(f"\n|00000> amplitude: {ghz5.state[0]:.3f}")
print(f"|11111> amplitude: {ghz5.state[-1]:.3f}")
print(f"All others: ~0.000")
```

Output:

```
5-qubit GHZ created!
State vector size: (32,) (2^5 = 32 dimensions)
Systems: 5
Entanglement entropy: 1.000

|00000> amplitude: 0.707
|11111> amplitude: 0.707
All others: ~0.000
```

The pattern holds: $|GHZ_5\rangle = \frac{|00000\rangle + |11111\rangle}{\sqrt{2}}$

8.8. Partial Measurements on GHZ States

What happens if you measure just **one** qubit from a GHZ state?

8. Understanding GHZ States

```
# Create GHZ state
program = QRLProgram("Partial GHZ Measurement")
q0 = program.create_system()
q1 = program.create_system()
q2 = program.create_system()
ghz = program.entangle(q0, q1, q2)

print("Before measurement:")
print(f"State: {ghz.state}")
print(f"Entropy: {ghz.entanglement_entropy:.3f}")

# Measure only the first qubit
question_z = create_question(QuestionType.SPIN_Z, subsystem=0)
alice = program.add_perspective("alice")
result = program.ask(ghz, question_z, perspective="alice")

print(f"\nMeasured qubit 0: {result[q0.system_id]}")
print(f"\nAfter measurement:")
print(f"State: {ghz.state}")
print(f"Entropy: {ghz.entanglement_entropy:.3f}")
```

Possible output (if measured 0):

Before measurement:
State: [0.707 0. 0. 0. 0. 0. 0. 0.707]
Entropy: 1.000

Measured qubit 0: 0

After measurement:
State: [1. 0. 0. 0. 0. 0. 0. 0.]
Entropy: 0.000

8.9. GHZ States and Quantum Non-Locality

Key insight: Measuring one qubit **collapses the entire GHZ state** to a product state ($|000\rangle$ or $|111\rangle$). The remaining qubits are **no longer entangled!**

This is different from some other entangled states (like W states, which we'll see in Chapter 11).

8.9. GHZ States and Quantum Non-Locality

GHZ states provide the **strongest violation of local realism** (even stronger than Bell states).

Consider this scenario:

1. Create a 3-qubit GHZ state
2. Send each qubit to Alice, Bob, and Charlie (far apart)
3. Each measures in X or Y basis
4. Compare results

Classical prediction (local realism): Certain combinations should occur

Quantum prediction (GHZ state): Some combinations **never** occur, violating local realism

This is the **GHZ paradox** - a starker demonstration of quantum non-locality than Bell's theorem.

8.10. Complete Working Example

Here's a complete program to explore GHZ states:

8. Understanding GHZ States

```
from qrl import QRLProgram, create_question, QuestionType
import numpy as np

def main():
    print("=" * 60)
    print("GHZ STATE DEMONSTRATION")
    print("=" * 60)

    # Create 3-qubit GHZ state
    program = QRLProgram("GHZ Demo")
    q0 = program.create_system()
    q1 = program.create_system()
    q2 = program.create_system()
    ghz = program.entangle(q0, q1, q2)

    # Display state information
    print(f"\nState vector: {ghz.state}")
    print(f"Entanglement entropy: {ghz.entanglement_entropy:.3f}")
    print(f"Number of systems: {len(ghz.systems)}")

    # Test correlations
    print("\n" + "=" * 60)
    print("TESTING 3-WAY CORRELATIONS (10 trials)")
    print("=" * 60)

    for trial in range(10):
        # Create fresh GHZ state for each trial
        p = QRLProgram(f"Trial {trial}")
        q0 = p.create_system()
        q1 = p.create_system()
        q2 = p.create_system()
        ghz_trial = p.entangle(q0, q1, q2)
```

8.11. Performance Note: State Vector Size

```
# Measure
question = create_question(QuestionType.SPIN_Z)
observer = p.add_perspective("observer")
result = p.ask(ghz_trial, question, perspective="observer")

outcomes = list(result.values())
outcome_str = ''.join(map(str, outcomes))

# Check if all match
all_match = "✓" if outcomes[0] == outcomes[1] == outcomes[2] else "x"

print(f"Trial {trial+1:2d}: {outcome_str} {all_match}")

print("\n✓ = All qubits match (expected behavior)")
print("x = Mixed outcomes (should never happen for GHZ)")

if __name__ == "__main__":
    main()
```

Save as `ghz_demo.py` and run:

```
python ghz_demo.py
```

8.11. Performance Note: State Vector Size

As you add more qubits, the state vector grows exponentially:

- 2 qubits: 4 complex numbers (32 bytes)
- 3 qubits: 8 complex numbers (64 bytes)
- 5 qubits: 32 complex numbers (256 bytes)
- 10 qubits: 1,024 complex numbers (8 KB)
- 20 qubits: 1,048,576 complex numbers (8 MB)

8. Understanding GHZ States

- **30 qubits: 1,073,741,824 complex numbers (8 GB!)**

For large systems, you'll need:

- **Tensor networks** (MPS/PEPS) - efficient approximations
- **Stabilizer formalism** - exact for certain states (including GHZ!)
- **MBQC compilation** - Convert to measurement patterns

We'll explore these in Part IV: Advanced Topics.

8.12. What You Learned

Congratulations! You now understand:

✓ **GHZ states** - Maximally entangled n-qubit states ✓ **Multipartite entanglement** - Genuine n-way correlations ✓ **Creating GHZ states** in QRL with `entangle(*qubits)` ✓ **Perfect correlations** - All qubits measure the same ✓ **Partial measurements** - Measuring one collapses all ✓ **Quantum non-locality** - Stronger than Bell states

8.13. Key Concepts

1. **GHZ state formula:** $|GHZ_n\rangle = \frac{|0^n\rangle + |1^n\rangle}{\sqrt{2}}$
2. **Perfect n-way correlation:** All qubits always measure the same
3. **Fragile to loss:** Losing one qubit destroys all entanglement
4. **Multipartite:** Can't be decomposed into Bell pairs
5. **Exponential scaling:** State vector size = 2^n complex numbers

8.14. Exercises

8.14.1. Exercise 1: Verify 4-Qubit GHZ

Create a 4-qubit GHZ state and verify:

- State vector has correct form
- All 4 qubits show perfect correlation (100 trials)
- Entropy is 1.0

8.14.2. Exercise 2: Cross-Basis GHZ Measurements

Measure a 3-qubit GHZ state in **X-basis** instead of Z-basis. Do correlations still hold? Why or why not?

Hint: Use `QuestionType.SPIN_X`

8.14.3. Exercise 3: GHZ State Distribution

Run 1000 trials of GHZ measurement. Count how many times you get:

- All 0s: 000
- All 1s: 111
- Mixed: anything else

Plot a histogram. What distribution do you expect?

8. Understanding GHZ States

8.14.4. Exercise 4: Sequential Partial Measurements

Start with 4-qubit GHZ. Measure qubits one by one:

1. Measure qubit 0 $\rightarrow$ observe state collapse
2. Measure qubit 1 $\rightarrow$ observe further collapse
3. Measure qubit 2 $\rightarrow$ what happens to entropy?
4. Measure qubit 3 $\rightarrow$ final state?

Track the entropy at each step.

8.14.5. Exercise 5: Large GHZ State

Create the largest GHZ state your computer can handle:

- Try 10, 15, 20 qubits
- Measure memory usage
- What's your practical limit?

Memory Warning

A 25-qubit state vector requires ~256 MB of RAM. A 30-qubit state needs ~8 GB. Be careful when scaling up!

8.15. Going Deeper

8.15.1. Theory: Why GHZ $\neq$ Bell Pairs

You might wonder: “Can I build a 3-qubit GHZ from Bell pairs?”

Answer: No!

GHZ states exhibit **genuine multipartite entanglement** that can't be created from bipartite (2-qubit) entanglement alone. This is a deep result in quantum information theory.

Mathematically: GHZ states belong to a different **entanglement class** under LOCC (Local Operations and Classical Communication).

8.15.2. Application: Quantum Secret Sharing

GHZ states enable **quantum secret sharing**:

- Alice has a secret bit
- Encodes it in a GHZ state shared with Bob and Charlie
- **Any 2 of the 3** can reconstruct the secret
- But **no single person** can learn it alone

This is impossible classically without trusted parties!

8.15.3. Connection to MBQC

GHZ states are **resource states** for measurement-based quantum computing:

- Create large GHZ state
- Measure qubits in chosen bases
- Measurement pattern implements quantum algorithm
- Output is measurement outcomes

We'll explore this in Part II: MBQC Theory.

8. Understanding GHZ States

Next Steps

Want to compare GHZ with other entangled states?

- **Chapter 11: W States** - Different kind of multipartite entanglement (robust to qubit loss!)
- **Chapter 5: Cluster States** - Even more qubits, different structure (MBQC resource states)
- **Chapter 13: Categorical QM** - String diagram representation of GHZ states

8.16. Summary

GHZ states are the gateway to multipartite quantum systems. You've learned:

- How to create them in QRL (just `entangle(*qubits)`)
- Their perfect n-way correlations
- How they differ from Bell states
- Their role in quantum non-locality
- How they scale (and when to worry about memory)

GHZ states are fundamental to quantum computing, quantum communication, and quantum sensing. Mastering them is essential for understanding advanced quantum protocols.

Next: [Chapter 9: Measurement in QRL](#) - Explore different measurement bases and contexts

9. Graph Extraction for MBQC

Now that you’ve created Bell states and GHZ states, it’s time to see how QRL converts these quantum relations into **graph states** for measurement-based quantum computing (MBQC). This is the first step in the MBQC compilation pipeline!

9.1. What is Graph Extraction?

In MBQC, quantum computations are performed by:

1. Preparing a **cluster state** (a special kind of graph state)
2. Measuring qubits in specific bases
3. Applying corrections based on measurement outcomes

Graph extraction is the process of converting a QRL `QuantumRelation` into a graph representation where:

- **Nodes** represent qubits (prepared in $|+\rangle$ state)
- **Edges** represent CZ (controlled-Z) entangling gates

9.1.1. Why Graph States Matter

Graph states are the foundation of MBQC:

$$|G\rangle = \prod_{(i,j) \in E} CZ_{ij} |+\rangle^{\otimes n}$$

9. Graph Extraction for MBQC

where E is the set of edges in graph G .

Key insight: Different entanglement structures produce different graph topologies!

9.2. Graph Topologies for Common States

Different quantum states have characteristic graph structures:

State	Topology	Nodes	Edges	Description
Bell	Edge graph	2	1	Two nodes connected by a single edge
GHZ	Star graph	n	n-1	Central node connected to all others
W	Ring graph	n	n	Cyclic connections forming a loop

Visual representation:

Bell State: 0 — 1

GHZ₃ State: 1
 |
 0 — 2

GHZ₄ State: 1 —
 2 —
 3 — 0

9.3. The Graph Extraction API

W State:

```
graph LR
    0 --- 1
    1 --- 2
    0 --- 2
```

9.3. The Graph Extraction API

QRL provides three main functions in the `qrl.mbqc` module:

9.3.1. 1. `extract_graph()` - Convert Relation to Graph

```
from qrl.mbqc import extract_graph

graph = extract_graph(relation)
```

Returns: A NetworkX Graph object with:

- Nodes: qubit indices (0, 1, 2, ...)
- Edges: CZ operations needed to create the cluster state
- Metadata: state type, number of qubits, description

9.3.2. 2. `analyze_entanglement_structure()` - Detect State Type

```
from qrl.mbqc import analyze_entanglement_structure

info = analyze_entanglement_structure(relation)
# Returns: {
#   'num_qubits': 2,
#   'state_type': 'bell',
```

9. Graph Extraction for MBQC

```
# 'entanglement_entropy': 1.0,  
# 'topology_hint': 'edge'  
# }
```

This function automatically detects whether you have a Bell state, GHZ state, W state, or unknown state.

9.3.3. 3. visualize_graph() - Display Graph Structure

```
from qrl.mbcc import visualize_graph  
  
print(visualize_graph(graph))  
# Shows nodes, edges, and adjacency information
```

9.4. Tutorial: Extract Bell State Graph

Let's start with the simplest case - extracting a graph from a Bell state:

```
from qrl import QRLProgram  
from qrl.mbcc import extract_graph, analyze_entanglement_structure, visualize_graph  
  
# Create Bell state  
program = QRLProgram("Bell State Graph Extraction")  
q0 = program.create_system()  
q1 = program.create_system()  
bell = program.entangle(q0, q1)  
  
print("Created Bell state:")  
print(f"State vector: {bell.state}")  
print(f"Entanglement entropy: {bell.entanglement_entropy:.3f}")
```


9.4. Tutorial: Extract Bell State Graph

Output:

```
Created Bell state:  
State vector: [0.707+0j  0.+0j  0.+0j  0.707+0j]  
Entanglement entropy: 1.000
```

Now let's analyze the entanglement structure:

```
# Analyze what kind of state this is  
info = analyze_entanglement_structure(bell)  
  
print("\nEntanglement analysis:")  
print(f"  Number of qubits: {info['num_qubits']}")  
print(f"  State type: {info['state_type']}")  
print(f"  Entanglement entropy: {info['entanglement_entropy']:.3f}")  
print(f"  Suggested topology: {info['topology_hint']}")
```

Output:

```
Entanglement analysis:  
  Number of qubits: 2  
  State type: bell  
  Entanglement entropy: 1.000  
  Suggested topology: edge
```

Finally, extract the graph:

```
# Extract graph representation  
graph = extract_graph(bell)  
  
print("\nExtracted graph:")  
print(visualize_graph(graph))
```

9. Graph Extraction for MBQC

Output:

```
Extracted graph:  
Graph: BELL state (2 qubits)  
Nodes (2): [0, 1]  
Edges (1): [(0, 1)]
```

```
Adjacency:  
  0: [1]  
  1: [0]
```

Perfect! Our Bell state produces an **edge graph** with 2 nodes and 1 edge, exactly as expected.

9.5. Tutorial: Extract GHZ State Graph

Now let's try a more complex state - a 3-qubit GHZ state:

```
# Create GHZ3 state  
program = QRLProgram("GHZ Graph Extraction")  
q0 = program.create_system()  
q1 = program.create_system()  
q2 = program.create_system()  
ghz3 = program.entangle(q0, q1, q2)  
  
# Analyze and extract  
info = analyze_entanglement_structure(ghz3)  
graph = extract_graph(ghz3)  
  
print(f"State type: {info['state_type']}")  
print(f"\n{visualize_graph(graph)}")
```

9.6. Scaling to 4+ Qubits

Output:

State type: ghz

Graph: GHZ state (3 qubits)

Nodes (3): [0, 1, 2]

Edges (2): [(0, 1), (0, 2)]

Adjacency:

0: [1, 2]

1: [0]

2: [0]

Notice the **star topology**: qubit 0 is the central node connected to both qubits 1 and 2. This is characteristic of GHZ states!

9.6. Scaling to 4+ Qubits

Let's verify that GHZ₄ also produces a star graph:

```
# Create GHZ4 state
program = QRLProgram("GHZ4 Graph")
qubits = [program.create_system() for _ in range(4)]
ghz4 = program.entangle(*qubits)

# Extract graph
graph = extract_graph(ghz4)
print(visualize_graph(graph))
```

Output:

9. Graph Extraction for MBQC

Graph: GHZ state (4 qubits)
Nodes (4): [0, 1, 2, 3]
Edges (3): [(0, 1), (0, 2), (0, 3)]

Adjacency:
0: [1, 2, 3]
1: [0]
2: [0]
3: [0]

Perfect! Qubit 0 is connected to all others - a star with 3 edges.

9.7. Working with the NetworkX Graph

The extracted graph is a standard NetworkX Graph object, so you can use all NetworkX functionality:

```
import networkx as nx

# Extract graph
graph = extract_graph(bell)

# Check if nodes are connected
print(f"Are qubits 0 and 1 connected? {graph.has_edge(0, 1)}")

# Get neighbors of a node
print(f"Neighbors of qubit 0: {list(graph.neighbors(0))}")

# Get degree of each node
for node in graph.nodes():
    print(f"Qubit {node} has degree {graph.degree(node)}")
```

9.8. Understanding the Graph State

```
# Access metadata
print(f"State type: {graph.graph['state_type']}")
print(f"Description: {graph.graph['description']}")
```

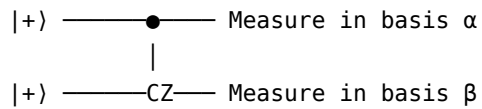
Output:

```
Are qubits 0 and 1 connected? True
Neighbors of qubit 0: [1]
Qubit 0 has degree 1
Qubit 1 has degree 1
State type: bell
Description: BELL state (2 qubits)
```

9.8. Understanding the Graph State

Why does a Bell state become an edge graph?

Circuit perspective:



State evolution:

1. Start: $|+\rangle \otimes |+\rangle$
2. Apply CZ between qubits 0-1
3. Result: Graph state with topology $0 \text{ --- } 1$

9. Graph Extraction for MBQC

Why this works:

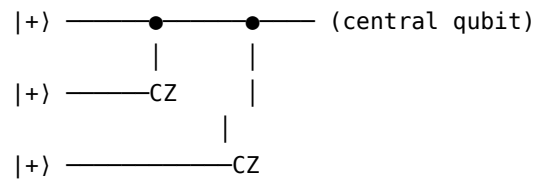
$$CZ_{01}|+\rangle|+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} = |\Phi^+\rangle$$

The CZ gate on $|+\rangle$ states creates the Bell state! The edge in the graph represents this CZ operation.

9.9. GHZ Star Topology Explained

For GHZ states, the star topology emerges from the preparation sequence:

Circuit for GHZ₃:



This creates edges (0,1) and (0,2) - exactly the star graph!

9.10. Complete Example: Analyze Multiple States

```
from qrl import QRLProgram
from qrl.mbqc import extract_graph, visualize_graph

def analyze_state(program, name, *qubits):
    """Helper to analyze any quantum state."""
    relation = program.entangle(*qubits)
```

9.11. What You Learned

```
graph = extract_graph(relation)

print(f"\n{'='*50}")
print(f"Analysis: {name}")
print('='*50)
print(visualize_graph(graph))
print(f"Graph density: {len(graph.edges()) / (len(graph.nodes()) * (len(graph.nodes())-1) / 2):.2f}")

# Main program
program = QRLProgram("Multi-State Analysis")

# Analyze Bell state
q0, q1 = program.create_system(), program.create_system()
analyze_state(program, "Bell State", q0, q1)

# Analyze GHZ3
q2, q3, q4 = program.create_system(), program.create_system(), program.create_system()
analyze_state(program, "GHZ3 State", q2, q3, q4)

# Analyze GHZ4
qubits = [program.create_system() for _ in range(4)]
analyze_state(program, "GHZ4 State", *qubits)
```

9.11. What You Learned

Congratulations! You now understand:

- ✓ **Graph extraction** - Converting QRL relations to graph states ✓
- State detection** - Automatic identification of Bell/GHZ/W states ✓
- Graph topologies** - Edge graphs, star graphs, and their meaning ✓
- MBQC connection** - How graphs represent cluster state preparation ✓
- NetworkX integration** - Working with extracted graphs

9.12. Key Takeaways

1. Different states $\rightarrow$ different graphs:

- Bell: edge graph (2 nodes, 1 edge)
- GHZ: star graph (n nodes, n-1 edges)
- W: ring graph (n nodes, n edges)

2. Edges = CZ gates:

- Each edge represents a CZ operation
- Applied to qubits in $|+\rangle$ state
- Creates the desired entanglement

3. Automatic detection:

- QRL analyzes the state vector
- Detects the state type
- Suggests appropriate topology

9.13. Exercises

9.13.1. Exercise 1: Bell State Verification

Create all four Bell states and verify they all produce edge graphs:

- $|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$ (default)
- $|\Phi^-\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$
- $|\Psi^+\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}}$
- $|\Psi^-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$

Do they all produce the same graph topology?

9.13.2. Exercise 2: GHZ Scaling

Create GHZ states with 2, 3, 4, and 5 qubits. For each:

- Extract the graph
- Count nodes and edges
- Verify the star topology
- Calculate graph density

Question: What pattern do you observe in the number of edges?

9.13.3. Exercise 3: Graph Properties

For a Bell state graph:

- Calculate the degree of each node
- Check if the graph is connected
- Compute the diameter of the graph
- List all possible paths between nodes

9.13.4. Exercise 4: W State Graph

Create a W state using `state_type="w"`:

```
w3 = program.entangle(q0, q1, q2, state_type="w")
```

- Extract the graph
- Compare its topology to GHZ₃
- How many edges does it have?
- What's the difference in structure?

9. Graph Extraction for MBQC

9.13.5. Exercise 5: Custom Analysis

Write a function that:

1. Takes a `QuantumRelation` as input
2. Extracts the graph
3. Returns a dictionary with:
 - Number of nodes
 - Number of edges
 - Average degree
 - Graph diameter
 - Is connected? (True/False)

Test it on Bell, GHZ, and W states.

9.14. Next Steps

You've mastered graph extraction! Next, you'll learn how to:

- Generate **measurement patterns** from graphs (Phase 2)
- Understand **adaptive corrections** for measurement outcomes
- Compile complete QRL programs to MBQC

Next: Chapter 10: W States and Entanglement Classes (coming soon)

See also:

- [Chapter 5: Cluster States](#) - Theoretical foundation
- [Chapter 6: Measurement Patterns](#) - What comes next

10. Measurement Pattern Generation

In the previous chapter, you learned how to extract graph states from quantum relations. Now you'll discover how to generate **measurement patterns** - the complete execution plans that tell an MBQC system what to measure and when.

10.1. What is a Measurement Pattern?

A **measurement pattern** is a complete specification for executing quantum computation using MBQC. It consists of five key components:

1. **Preparation:** Which qubits to prepare in $|+\rangle$ state
2. **Entanglement:** Which CZ gates to apply (creating the cluster state)
3. **Measurements:** What basis and angle to measure each qubit
4. **Corrections:** Adaptive Pauli corrections based on measurement outcomes
5. **Output:** Which qubits contain the final result

10.1.1. Pattern Structure

```
from qrl.mbgc import MeasurementPattern, Measurement, Correction

pattern = MeasurementPattern(
    preparation=[0, 1],          # Prepare qubits 0 and 1 in  $|+\rangle$ 
```

10. Measurement Pattern Generation

```
    entanglement=[(0, 1)],      # Apply CZ gate between them
    measurements=[...],          # List of measurements
    corrections=[...],           # List of adaptive corrections
    output_qubits=[0, 1],        # Output on qubits 0 and 1
    description="Pattern description"
)
```

10.2. The Pattern Generation API

QRL provides several functions for generating measurement patterns:

Function	Purpose
<code>generate_bell_state_pattern()</code>	Bell state preparation
<code>generate_ghz_state_pattern(n)</code>	n-qubit GHZ state
<code>generate_single_qubit_gate_pattern(gate)</code>	H, X, Z, S, T gates
<code>generate_rotation_pattern(axis, angle)</code>	Arbitrary rotations
<code>generate_cnot_pattern()</code>	CNOT (controlled-X) gate
<code>generate_cz_pattern()</code>	CZ (controlled-Z) gate
<code>generate_pattern_from_relation(rel)</code>	From QuantumRelation
<code>combine_patterns(p1, p2)</code>	Sequential composition

10.3. Tutorial: Bell State Pattern

The simplest pattern is Bell state preparation:

```
from qrl.mbc import generate_bell_state_pattern

# Generate pattern for Bell state
pattern = generate_bell_state_pattern()
```

10.4. Tutorial: GHZ State Pattern

```
print(pattern)
```

Output:

```
Pattern: Bell state  $|\Phi^+\rangle$  preparation
Qubits: 2
Preparation: 2 qubits in  $|+\rangle$ 
Entanglement: 1 CZ gates
Measurements: 0
Corrections: 0
Output qubits: [0, 1]
Measurement depth: 0
```

10.3.1. Understanding the Pattern

This pattern tells us:

- **Prepare** 2 qubits in $|+\rangle$: preparation=[0, 1]
- **Apply** 1 CZ gate: entanglement=[(0, 1)]
- **No measurements** needed (this IS the prepared state)
- **Output** both qubits: output_qubits=[0, 1]

Physical interpretation:

$$CZ_{01}|+\rangle_0|+\rangle_1 = \frac{|00\rangle + |11\rangle}{\sqrt{2}} = |\Phi^+\rangle$$

10.4. Tutorial: GHZ State Pattern

Scaling to multiple qubits is straightforward:

10. Measurement Pattern Generation

```
from qrl.mbc import generate_ghz_state_pattern

# Generate GHZ3 pattern
ghz3_pattern = generate_ghz_state_pattern(3)

print(ghz3_pattern)
print(f"\nEntanglement edges: {ghz3_pattern.entanglement}")
```

Output:

```
Pattern: GHZ_3 state preparation
Qubits: 3
Preparation: 3 qubits in |+)
Entanglement: 2 CZ gates
Measurements: 0
Corrections: 0
Output qubits: [0, 1, 2]
Measurement depth: 0
```

```
Entanglement edges: [(0, 1), (0, 2)]
```

The **star topology** is visible in the edges: qubit 0 is the center, connected to qubits 1 and 2.

10.5. Tutorial: Single-Qubit Gates

Now we enter the realm of actual computation! Single-qubit gates are implemented via measurement:

10.5.1. Hadamard Gate

```
from qrl.mbgc import generate_single_qubit_gate_pattern

# Generate pattern for H gate
h_pattern = generate_single_qubit_gate_pattern("H", input_qubit=0)

print(h_pattern)
print(f"\nMeasurement details:")
for m in h_pattern.measurements:
    print(f"  Qubit {m.qubit}: angle={m.angle:.3f}, plane={m.plane}")
```

Output:

```
Pattern: Hadamard gate
Qubits: 2
Preparation: 2 qubits in |+>
Entanglement: 1 CZ gates
Measurements: 1
Corrections: 1
Output qubits: [0]
Measurement depth: 1

Measurement details:
  Qubit 1: angle=0.000, plane=XY
```

Key insight: The H gate uses an **ancilla qubit** (qubit 1) which gets measured in the XY plane at angle 0. The result appears on qubit 0!

10. Measurement Pattern Generation

10.5.2. Pauli Gates

```
# X gate
x_pattern = generate_single_qubit_gate_pattern("X", input_qubit=0)
print(f"X gate: measure at angle  $\pi$  in plane {x_pattern.measurements[0].plane}")

# Z gate
z_pattern = generate_single_qubit_gate_pattern("Z", input_qubit=0)
print(f"Z gate: measure at angle 0 in plane {z_pattern.measurements[0].plane}")
```

Output:

X gate: measure at angle 3.142 in plane XY
Z gate: measure at angle 0.000 in plane XZ

Pattern: Different gates use different measurement planes and angles!

10.6. Understanding Measurement Angles

The measurement angle determines the gate's action:

Gate	Plane	Angle	Effect
H	XY	0	Superposition
X	XY	π	Bit flip
Z	XZ	0	Phase flip
S	XZ	$\pi/2$	$\sqrt{Z}$ gate
T	XZ	$\pi/4$	$\sqrt{S}$ gate

The angle θ determines the measurement basis:

$$|\pm_\theta\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm e^{i\theta}|1\rangle)$$

10.7. Tutorial: Arbitrary Rotations

For custom rotations, use `generate_rotation_pattern`:

```
from qrl.mbgc import generate_rotation_pattern
import numpy as np

# Rotation around X axis by  $\pi/4$ 
rx_pattern = generate_rotation_pattern("X", np.pi/4, input_qubit=0)

print(f"R_X( $\pi/4$ ) pattern:")
print(rx_pattern)
print(f"Measurement: angle={rx_pattern.measurements[0].angle:.3f}, plane={rx_pattern.measurements[0].plane}")
```

Output:

```
R_X( $\pi/4$ ) pattern:
Pattern: R_X(0.785) rotation
Qubits: 2
Measurement: angle=0.785, plane=YZ
```

Axis mapping:

- X rotation $\rightarrow$ YZ plane
- Y rotation $\rightarrow$ XZ plane
- Z rotation $\rightarrow$ XY plane

10.8. Tutorial: Two-Qubit Gates

Two-qubit gates are essential for universal quantum computation. MBQC implements them using cluster states with multiple measurements.

10. Measurement Pattern Generation

10.8.1. CZ Gate (Controlled-Z)

The CZ gate is **native** to MBQC - it's simply an edge in the cluster state graph!

```
from qrl.mbgc import generate_cz_pattern

# Generate CZ pattern
cz_pattern = generate_cz_pattern()

print(cz_pattern)
```

Output:

```
Pattern: CZ gate
Qubits: 2
Preparation: 2 qubits in |+>
Entanglement: 1 CZ gates
Measurements: 0
Corrections: 0
Output qubits: [0, 1]
Measurement depth: 0
```

Key insight: CZ requires no measurements! It's just entanglement between qubits - the fundamental operation that builds cluster states.

Physical interpretation:

$$CZ_{01}|+\rangle_0|+\rangle_1 = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$$

10.8.2. CNOT Gate (Controlled-X)

CNOT is more complex - it requires a **4-qubit linear cluster** with measurements on two ancilla qubits:

```
from qrl.mbgc import generate_cnot_pattern

# Generate CNOT pattern
cnot_pattern = generate_cnot_pattern()

print(cnot_pattern)
print(f"\nEntanglement (linear cluster): {cnot_pattern.entanglement}")
print(f"Measurements: {len(cnot_pattern.measurements)}")
print(f"Corrections: {len(cnot_pattern.corrections)}")
```

Output:

```
Pattern: CNOT gate
Qubits: 4
Preparation: 4 qubits in |+>
Entanglement: 3 CZ gates
Measurements: 2
Corrections: 3
Output qubits: [0, 3]
Measurement depth: 2
```

```
Entanglement (linear cluster): [(0, 1), (1, 2), (2, 3)]
Measurements: 2
Corrections: 3
```

10.8.3. CNOT Structure Explained

The CNOT pattern uses this linear cluster topology:

10. Measurement Pattern Generation

control_in (0) --- ancilla_c (1) --- ancilla_t (2) --- target_in (3)

How it works:

1. **Preparation:** All 4 qubits start in $|+\rangle$
2. **Entanglement:** Apply CZ gates to create the linear cluster
3. **Measurements:**
 - Measure ancilla_c (qubit 1) in XY plane at angle 0
 - Measure ancilla_t (qubit 2) in XY plane at angle 0 (adaptive)
4. **Corrections:** Apply Pauli corrections based on measurement outcomes
5. **Output:** Qubits 0 (control) and 3 (target) contain the result

```
# Examine CNOT measurements
for i, m in enumerate(cnot_pattern.measurements):
    print(f"Measurement {i+1}:")
    print(f"  Qubit: {m.qubit}")
    print(f"  Angle: {m.angle}")
    print(f"  Plane: {m.plane}")
    print(f"  Adaptive: {m.adaptive}")
    print(f"  Depends on: {m.depends_on}")
```

Output:

```
Measurement 1:
  Qubit: 1
  Angle: 0.0
  Plane: XY
  Adaptive: False
  Depends on: []
Measurement 2:
  Qubit: 2
```

10.8. Tutorial: Two-Qubit Gates

Angle: 0.0
Plane: XY
Adaptive: True
Depends on: [1]

Important: The second measurement is **adaptive** - its effective angle depends on the first measurement's outcome!

10.8.4. Why CNOT Needs 4 Qubits

Unlike in the circuit model where CNOT acts directly on 2 qubits, MBQC uses a “teleportation-like” structure:

Gate	Circuit Qubits	MBQC Qubits	Measurements	Depth
CZ	2	2	0	0
CNOT	2	4	2	2
H	1	2	1	1

This is the **space-time tradeoff** of MBQC: we use more qubits but can parallelize operations.

10.8.5. Universal Quantum Computation

With CNOT and single-qubit rotations, you have a **universal gate set**! Any quantum circuit can be compiled to MBQC:

```
from qrl.mbgc import (  
    generate_single_qubit_gate_pattern,  
    generate_cnot_pattern,  
    combine_patterns  
)
```

10. Measurement Pattern Generation

```
# Build a simple quantum circuit: H on qubit 0, then CNOT
h_pattern = generate_single_qubit_gate_pattern("H", 0)
cnot_pattern = generate_cnot_pattern()

# This creates a Bell state via circuit!
circuit_pattern = combine_patterns(h_pattern, cnot_pattern)

print(f"H + CNOT circuit:")
print(f"  Total qubits: {circuit_pattern.num_qubits}")
print(f"  Total measurements: {len(circuit_pattern.measurements)}")
print(f"  Measurement depth: {circuit_pattern.measurement_depth}")
```

10.9. Tutorial: Pattern from Relation

You can generate patterns directly from existing quantum relations:

```
from qrl import QRLProgram
from qrl.mbqc import generate_pattern_from_relation

# Create a Bell state
program = QRLProgram("Pattern from Relation")
q0 = program.create_system()
q1 = program.create_system()
bell = program.entangle(q0, q1)

# Generate pattern from the relation
pattern = generate_pattern_from_relation(bell)

print("Generated pattern:")
print(pattern)
```

This combines graph extraction and pattern generation in one step!

10.10. Adaptive Corrections

Measurement patterns include **corrections** - Pauli operations that depend on measurement outcomes:

```
h_pattern = generate_single_qubit_gate_pattern("H")

print(f"Number of corrections: {len(h_pattern.corrections)}")
for corr in h_pattern.corrections:
    print(f"  Target: qubit {corr.target}")
    print(f"  Type: {corr.correction_type}")
    print(f"  Depends on: {corr.depends_on}")
```

Output:

```
Number of corrections: 1
  Target: qubit 0
  Type: Z
  Depends on: [1]
```

Meaning: If measuring qubit 1 gives outcome “1”, apply Z to qubit 0.

Why needed: Measurements are probabilistic. Corrections ensure deterministic results.

10. Measurement Pattern Generation

10.11. Pattern Combination

Combine patterns sequentially to build circuits:

```
from qrl.mbgc import combine_patterns

# Create H gate followed by X gate
h_pattern = generate_single_qubit_gate_pattern("H", 0)
x_pattern = generate_single_qubit_gate_pattern("X", 0)

# Combine them
hx_pattern = combine_patterns(h_pattern, x_pattern)

print(f"Combined pattern:")
print(hx_pattern)
print(f"Total qubits: {hx_pattern.num_qubits}")
print(f"Total measurements: {len(hx_pattern.measurements)}")
```

Output:

```
Combined pattern:
Pattern: Hadamard gate + Pauli X gate
Total qubits: 4
Total measurements: 2
```

The combined pattern has 4 qubits (2 from each gate) and 2 measurements.

10.12. Measurement Depth

The **measurement depth** is the longest chain of dependent measurements:

10.13. Complete Example: Analyze Multiple Patterns

```
bell_pattern = generate_bell_state_pattern()
print(f"Bell state depth: {bell_pattern.measurement_depth}") # 0

h_pattern = generate_single_qubit_gate_pattern("H")
print(f"H gate depth: {h_pattern.measurement_depth}") # 1

combined = combine_patterns(h_pattern, generate_single_qubit_gate_pattern("X"))
print(f"H + X depth: {combined.measurement_depth}") # 2
```

Lower depth = more parallelism - measurements can happen simultaneously if independent.

10.13. Complete Example: Analyze Multiple Patterns

```
from qrl.mbc import (
    generate_bell_state_pattern,
    generate_ghz_state_pattern,
    generate_single_qubit_gate_pattern,
    generate_rotation_pattern
)
import numpy as np

def analyze_pattern(name, pattern):
    """Helper to analyze any measurement pattern."""
    print(f"\n{'='*50}")
    print(f"{name}")
    print(f"{'='*50}")
    print(f"Qubits: {pattern.num_qubits}")
    print(f"CZ gates: {len(pattern.entanglement)}")
    print(f"Measurements: {len(pattern.measurements)}")
```

10. Measurement Pattern Generation

```
print(f"Corrections: {len(pattern.corrections)}")
print(f"Depth: {pattern.measurement_depth}")

# Analyze various patterns
analyze_pattern("Bell State", generate_bell_state_pattern())
analyze_pattern("GHZ State", generate_ghz_state_pattern(3))
analyze_pattern("Hadamard Gate", generate_single_qubit_gate_pattern("H"))
analyze_pattern("R_Y( $\pi/3$ )", generate_rotation_pattern("Y", np.pi/3))
```

10.14. What You Learned

Congratulations! You now understand:

✓ **Measurement patterns** - Complete MBQC execution plans ✓ **Pattern components** - Preparation, entanglement, measurements, corrections
✓ **State preparation** - Bell and GHZ patterns ✓ **Single-qubit gates** - H, X, Z, S, T via measurement ✓ **Two-qubit gates** - CZ (native) and CNOT (4-qubit cluster) ✓ **Rotation patterns** - Arbitrary angle rotations
✓ **Pattern combination** - Building quantum circuits ✓ **Universal computation** - CNOT + rotations = any quantum circuit ✓ **Measurement depth** - Parallelism analysis

10.15. Key Takeaways

1. MBQC = Preparation + Measurement:

- Prepare cluster state
- Measure qubits in specific bases
- Apply corrections

2. Gates become measurements:

- Single-qubit gates use 1 ancilla qubit
- CNOT uses 2 ancilla qubits (4 total)
- Measurement angle determines the gate
- Corrections ensure deterministic output

3. Two-qubit gates:

- CZ is native (just an edge in the graph)
- CNOT requires a linear 4-qubit cluster
- CNOT + single-qubit gates = universal computation

4. Patterns are composable:

- Combine patterns to build circuits
- Sequential composition via `combine_patterns`
- Measurement depth shows parallelism

10.16. Exercises

10.16.1. Exercise 1: Pattern Analysis

Generate patterns for all Pauli gates (X, Y, Z) and compare:

- Measurement angles
- Measurement planes
- Correction types

What pattern do you notice?

10.16.2. Exercise 2: Measurement Depth

Create a pattern that combines three gates: H, S, T.

- What's the measurement depth?

10. Measurement Pattern Generation

- How many qubits are needed?
- How many measurements total?

10.16.3. Exercise 3: Custom Rotation

Generate a pattern for $R_Z(\pi/8)$ and verify:

- The measurement plane is XY
- The angle is $\pi/8$
- There's 1 correction

10.16.4. Exercise 4: GHZ Scaling

Generate GHZ patterns for $n = 2, 3, 4, 5$ qubits. For each, calculate:

- Number of CZ gates
- Graph density (edges / possible edges)
- Resource efficiency (qubits vs entanglement)

10.16.5. Exercise 5: Pattern Composer

Write a function that:

1. Takes a list of gate names: ["H", "X", "S"]
2. Generates a pattern for each gate
3. Combines them sequentially
4. Returns the combined pattern

Test it with different gate sequences.

10.16.6. Exercise 6: Pattern Validator

Write a function that checks if a pattern is valid:

- All measured qubits are prepared
- All entanglement uses prepared qubits
- No circular dependencies in measurements
- Corrections only depend on earlier measurements

10.16.7. Exercise 7: CNOT Analysis

Analyze the CNOT pattern in detail:

```
from qrl.mbc import generate_cnot_pattern

cnot = generate_cnot_pattern()
```

1. Draw the 4-qubit linear cluster graph
2. List all corrections and what they depend on
3. Verify: if both measurements give outcome 0, what corrections are applied?
4. Verify: if both measurements give outcome 1, what corrections are applied?

10.16.8. Exercise 8: Universal Circuit

Build the pattern for a simple quantum circuit that creates a Bell state:

1. Apply H to qubit 0
2. Apply CNOT with control=0, target=1

Compare the total resources (qubits, measurements, depth) with `generate_bell_state_pattern()`. Why is the direct pattern more efficient?

10. Measurement Pattern Generation

10.16.9. Exercise 9: CZ vs CNOT

Compare CZ and CNOT patterns:

- Resource usage (qubits, measurements)
- Why is CZ “native” but CNOT is not?
- How would you implement CNOT using CZ and single-qubit gates?

10.17. Next Steps

You’ve mastered pattern generation! Next, you’ll learn about:

- **Adaptive corrections** - Implementing quantum teleportation (Phase 3)
- **Pattern execution** - Simulating MBQC (Phase 4)
- **Photonic compilation** - Real hardware backends (Future)

Next: Chapter 11: Adaptive Corrections and Quantum Teleportation (coming soon)

See also:

- [Chapter 9: Graph Extraction](#) - Foundation for patterns
- [Chapter 6: Measurement Patterns Theory](#) - Theoretical background

11. Adaptive Corrections and Quantum Teleportation

In the previous chapter, you learned how to generate measurement patterns for quantum gates. Now you'll discover **adaptive corrections** - the mechanism that makes MBQC deterministic despite probabilistic measurements. We'll use **quantum teleportation** as our key example.

11.1. Why Adaptive Corrections?

In MBQC, measurements are inherently probabilistic - each measurement randomly returns 0 or 1. Without corrections, your quantum computation would give random results!

The solution: Apply Pauli corrections (X, Z, or both) based on earlier measurement outcomes. This “feedforward” of classical information makes the computation deterministic.

11.1.1. The MBQC Correction Principle

Measurement outcome = 0 → No correction needed
Measurement outcome = 1 → Apply Pauli correction

This is why MBQC requires **classical communication** alongside quantum operations.

11. Adaptive Corrections and Quantum Teleportation

11.2. The Adaptive Corrections API

QRL provides functions for working with corrections:

Function	Purpose
<code>apply_pauli_correction(state, qubit, type)</code>	Apply X, Z, or XZ correction
<code>compute_corrections(pattern, outcomes)</code>	Determine corrections from measurements
<code>generate_teleportation_pattern()</code>	Teleportation measurement pattern
<code>simulate_teleportation(state)</code>	Full teleportation simulation
<code>verify_teleportation_fidelity(in, out)</code>	Check teleportation quality
<code>correction_truth_table(n)</code>	All correction scenarios

11.3. Tutorial: Pauli Corrections

The three Pauli operators form the basis of all corrections:

11.3.1. X Correction (Bit Flip)

```
import numpy as np
from qrl.mbc import apply_pauli_correction

# |0> state
state_0 = np.array([1, 0], dtype=complex)

# Apply X correction
```


11.3. Tutorial: Pauli Corrections

```
corrected = apply_pauli_correction(state_0, 0, "X")

print(f"Before X: {state_0}")
print(f"After X: {corrected}")
```

Output:

```
Before X: [1.+0.j 0.+0.j]
After X:  [0.+0.j 1.+0.j]
```

X flips $|0\rangle \leftrightarrow |1\rangle$.

11.3.2. Z Correction (Phase Flip)

```
#  $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ 
state_plus = np.array([1, 1], dtype=complex) / np.sqrt(2)

# Apply Z correction
corrected = apply_pauli_correction(state_plus, 0, "Z")

print(f"Before Z: {state_plus}")
print(f"After Z: {corrected}")
```

Output:

```
Before Z: [0.707+0.j 0.707+0.j]
After Z:  [0.707+0.j -0.707+0.j]
```

Z transforms $|+\rangle \rightarrow |-\rangle$ by flipping the phase of $|1\rangle$.

11. Adaptive Corrections and Quantum Teleportation

11.3.3. Combined XZ Correction

```
# Apply both X then Z
state_0 = np.array([1, 0], dtype=complex)
corrected = apply_pauli_correction(state_0, 0, "XZ")

print(f"Before XZ: {state_0}")
print(f"After XZ: {corrected}")
# X:  $|0\rangle \rightarrow |1\rangle$ , then Z:  $|1\rangle \rightarrow -|1\rangle$ 
```

Output:

```
Before XZ: [1.+0.j 0.+0.j]
After XZ:  [0.+0.j -1.+0.j]
```

11.4. Quantum Teleportation

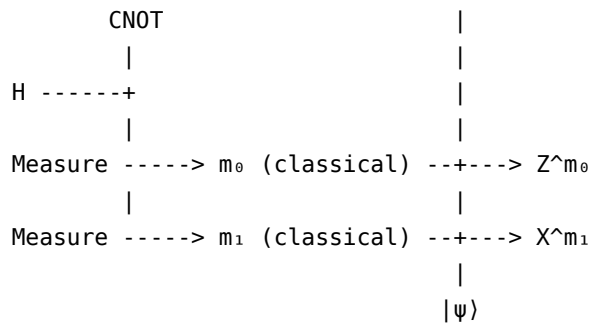
Quantum teleportation is the **canonical example** of adaptive corrections. It transfers a quantum state using only:

- A shared Bell pair (entanglement)
- Two classical bits (measurement outcomes)
- Pauli corrections

11.4.1. The Teleportation Protocol

Alice		Bob
ψ $\rangle$	-----+	

11.4. Quantum Teleportation



Steps:

1. Alice has input state $|\psi\rangle$ on qubit A
2. Alice and Bob share Bell pair $|\Phi^+\rangle$ on qubits B, C
3. Alice applies CNOT(A $\rightarrow$ B) then H(A)
4. Alice measures qubits A and B, gets outcomes m_0 and m_1
5. Bob applies $Z^{m_0}X^{m_1}$ to qubit C
6. Bob's qubit C now contains $|\psi\rangle$!

11.4.2. The Teleportation Pattern

```
from qrl.mbgc import generate_teleportation_pattern

pattern = generate_teleportation_pattern()

print(pattern)
```

Output:

```
Pattern: Quantum Teleportation
Qubits: 3
```

11. Adaptive Corrections and Quantum Teleportation

Preparation: 3 qubits in $|+\rangle$
Entanglement: 1 CZ gates
Measurements: 2
Corrections: 2
Output qubits: [2]
Measurement depth: 1

11.4.3. Understanding the Pattern Structure

```
# Examine pattern details
print(f"Preparation: qubits {pattern.preparation}")
print(f"Entanglement: {pattern.entanglement}")
print(f"Output: qubit {pattern.output_qubits}")

print(f"\nMeasurements:")
for m in pattern.measurements:
    print(f"  Qubit {m.qubit}: plane={m.plane}, angle={m.angle}")

print(f"\nCorrections:")
for c in pattern.corrections:
    print(f"  Target qubit {c.target}: type={c.correction_type}")
```

Output:

Preparation: qubits [0, 1, 2]
Entanglement: [(1, 2)]
Output: qubit [2]

Measurements:
 Qubit 0: plane=XY, angle=0.0
 Qubit 1: plane=XY, angle=0.0

11.5. Tutorial: Simulating Teleportation

Corrections:

Target qubit 2: type=Z

Target qubit 2: type=X

11.5. Tutorial: Simulating Teleportation

Let's teleport actual quantum states!

11.5.1. Teleporting Basis States

```
from qrl.mbgc import simulate_teleportation, verify_teleportation_fidelity

# Teleport  $|0\rangle$ 
input_state = np.array([1, 0], dtype=complex)
output_state, outcomes, corrections = simulate_teleportation(input_state)

print(f"Input: {input_state}")
print(f"Output: {output_state}")
print(f"Measurements: {outcomes}")
print(f"Corrections: {corrections}")

fidelity = verify_teleportation_fidelity(input_state, output_state)
print(f"Fidelity: {fidelity:.6f}")
```

Output:

Input: [1.+0.j 0.+0.j]

Output: [1.+0.j 0.+0.j]

Measurements: {0: 1, 1: 0}

11. Adaptive Corrections and Quantum Teleportation

Corrections: ['Z']

Fidelity: 1.000000

11.5.2. Teleporting Superposition States

```
# Teleport  $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ 
input_plus = np.array([1, 1], dtype=complex) / np.sqrt(2)
output, outcomes, corrections = simulate_teleportation(input_plus)

print(f"Input  $|+\rangle$ : {input_plus}")
print(f"Output: {output}")
print(f"Fidelity: {verify_teleportation_fidelity(input_plus, output):.6f}")
```

Output:

Input $|+\rangle$: [0.707+0.j 0.707+0.j]

Output: [0.707+0.j 0.707+0.j]

Fidelity: 1.000000

11.5.3. Teleporting Complex States

```
# Teleport  $|i+\rangle = (|0\rangle + i|1\rangle)/\sqrt{2}$ 
input_iplus = np.array([1, 1j], dtype=complex) / np.sqrt(2)
output, outcomes, corrections = simulate_teleportation(input_iplus)

print(f"Input  $|i+\rangle$ : {input_iplus}")
print(f"Output: {output}")
print(f"Fidelity: {verify_teleportation_fidelity(input_iplus, output):.6f}")
```

Output:

11.6. The Correction Truth Table

Input $|i+\rangle$: $[0.707+0.j \ 0.+0.707j]$
Output: $[0.707+0.j \ 0.+0.707j]$
Fidelity: 1.000000

Key Result: Teleportation achieves **perfect fidelity** for any input state!

11.6. The Correction Truth Table

Different measurement outcomes require different corrections:

```
from qrl.mbgc import correction_truth_table

table = correction_truth_table(n_measurements=2)

print("Teleportation Correction Truth Table")
print("=" * 40)
print(f"{'m0':<6} {'m1':<6} {'Corrections':<20}")
print("-" * 40)

for entry in table:
    m0, m1 = entry['outcomes']
    corr = ', '.join(entry['corrections']) if entry['corrections'] else "None"
    print(f"{'m0':<6} {'m1':<6} {'corr':<20}")
```

Output:

```
Teleportation Correction Truth Table
=====
m0      m1      Corrections
-----
```

11. Adaptive Corrections and Quantum Teleportation

0	0	None
0	1	X
1	0	Z
1	1	Z, X

11.6.1. Mathematical Explanation

After Alice's Bell measurement, Bob's qubit is in one of four states:

Outcome	Bob's State	Correction	Result
00	$\alpha 0\rangle + \beta 1\rangle$	None	$ \psi\rangle$
01	$\alpha 1\rangle + \beta 0\rangle$	X	$ \psi\rangle$
10	$\alpha 0\rangle - \beta 1\rangle$	Z	$ \psi\rangle$
11	$\alpha 1\rangle - \beta 0\rangle$	XZ	$ \psi\rangle$

Each outcome occurs with probability $1/4$, and the appropriate correction always recovers $|\psi\rangle$.

11.7. Computing Corrections from Patterns

Use `compute_corrections` to determine what corrections to apply:

```
from qrl.mbgc import generate_teleportation_pattern, compute_corrections

pattern = generate_teleportation_pattern()

# Test different measurement scenarios
scenarios = [
    {0: 0, 1: 0}, # Both measure 0
    {0: 1, 1: 0}, # First measures 1
```


11.8. Multi-Qubit Corrections

```
{0: 0, 1: 1}, # Second measures 1
{0: 1, 1: 1}, # Both measure 1
]

for outcomes in scenarios:
    corrections = compute_corrections(pattern, outcomes)
    types = [c['type'] for c in corrections]
    print(f"Outcomes {outcomes} → Corrections: {types if types else 'None'}")
```

Output:

```
Outcomes {0: 0, 1: 0} → Corrections: None
Outcomes {0: 1, 1: 0} → Corrections: ['Z']
Outcomes {0: 0, 1: 1} → Corrections: ['X']
Outcomes {0: 1, 1: 1} → Corrections: ['Z', 'X']
```

11.8. Multi-Qubit Corrections

Corrections can target specific qubits in multi-qubit systems:

```
# |00> state
state_00 = np.array([1, 0, 0, 0], dtype=complex)

# Apply X to second qubit only
corrected = apply_pauli_correction(state_00, qubit_idx=1, correction_type="X")

print(f"Before: {state_00} (|00>)")
print(f"After: {corrected} (|01>)")
```

Output:

11. Adaptive Corrections and Quantum Teleportation

Before: $[1.+0.j \ 0.+0.j \ 0.+0.j \ 0.+0.j]$ ($|00\rangle$)
After: $[0.+0.j \ 1.+0.j \ 0.+0.j \ 0.+0.j]$ ($|01\rangle$)

11.9. Complete Example: Teleportation Analysis

```
import numpy as np
from qrl.mbc import (
    simulate_teleportation,
    verify_teleportation_fidelity,
    generate_teleportation_pattern
)

def analyze_teleportation(name, input_state):
    """Analyze teleportation of a quantum state."""
    print(f"\n{'='*50}")
    print(f"Teleporting: {name}")
    print(f"{'='*50}")

    # Normalize input
    input_state = input_state / np.linalg.norm(input_state)

    # Run teleportation
    output, outcomes, corrections = simulate_teleportation(input_state)
    fidelity = verify_teleportation_fidelity(input_state, output)

    print(f"Input state: {input_state}")
    print(f"Output state: {output}")
    print(f"Measurements: m0={outcomes[0]}, m1={outcomes[1]}")
    print(f"Corrections: {corrections if corrections else 'None'}")
    print(f"Fidelity: {fidelity:.6f}")
```

11.10. What You Learned

```
# Verify probabilities preserved
in_probs = np.abs(input_state)**2
out_probs = np.abs(output)**2
print(f"P(0): {in_probs[0]:.4f} → {out_probs[0]:.4f}")
print(f"P(1): {in_probs[1]:.4f} → {out_probs[1]:.4f}")

return fidelity

# Test various states
states = [
    ("|0>", np.array([1, 0])),
    ("|1>", np.array([0, 1])),
    ("|+>", np.array([1, 1])),
    ("|>", np.array([1, -1])),
    ("|i>", np.array([1, 1j])),
    ("custom", np.array([0.6, 0.8])),
]

fidelities = []
for name, state in states:
    f = analyze_teleportation(name, state.astype(complex))
    fidelities.append(f)

print(f"\n{'='*50}")
print(f"Average Fidelity: {np.mean(fidelities):.6f}")
print(f"All states teleported successfully!")
print(f"{'='*50}")
```

11.10. What You Learned

Congratulations! You now understand:

11. Adaptive Corrections and Quantum Teleportation

- **Adaptive corrections** - Pauli operations conditioned on measurements
- **X correction** - Bit flip ($|0\rangle \leftrightarrow |1\rangle$)
- **Z correction** - Phase flip ($|+\rangle \leftrightarrow |-\rangle$)
- **Quantum teleportation** - State transfer via entanglement + classical bits
- **Teleportation pattern** - 3 qubits, 2 measurements, 2 corrections
- **Perfect fidelity** - Corrections ensure exact state transfer
- **Truth tables** - All correction scenarios mapped

11.11. Key Takeaways

1. **Measurements are probabilistic, corrections make them deterministic:**
 - Outcome 0 $\rightarrow$ usually no correction
 - Outcome 1 $\rightarrow$ apply Pauli correction
2. **Teleportation proves MBQC works:**
 - Transfers arbitrary quantum states
 - Uses only entanglement + classical communication
 - Perfect fidelity with correct corrections
3. **Corrections are the “feedforward” in MBQC:**
 - Classical information flows forward
 - Later operations depend on earlier measurements
 - This is what makes MBQC universal
4. **The correction truth table:**

m_0	m_1	Correction
0	0	None
0	1	X
1	0	Z
1	1	XZ

11.12. Exercises

11.12.1. Exercise 1: Pauli Algebra

Verify these Pauli identities using `apply_pauli_correction`:

- $X^2 = I$ (applying X twice returns to original)
- $Z^2 = I$
- $XZ = -ZX$ (they anti-commute)

11.12.2. Exercise 2: Teleportation Statistics

Run teleportation 100 times on the same input state.

- What fraction of runs require no correction?
- What fraction require X only? Z only? Both?
- Do these match the expected 25% each?

11.12.3. Exercise 3: Phase States

Teleport these phase states and verify fidelity:

- $|+i\rangle = (|0\rangle + i|1\rangle)/\sqrt{2}$
- $|-i\rangle = (|0\rangle - i|1\rangle)/\sqrt{2}$
- $e^{i\pi/4}|0\rangle + e^{-i\pi/4}|1\rangle$

11.12.4. Exercise 4: Correction Composition

If you need to apply X then Z:

- What single Pauli is equivalent?
- Verify using `apply_pauli_correction`
- Does the order matter (XZ vs ZX)?

11. Adaptive Corrections and Quantum Teleportation

11.12.5. Exercise 5: Multi-Round Teleportation

Implement “teleportation relay”:

1. Teleport from Alice to Bob
2. Bob teleports to Charlie
3. Verify Charlie has the original state

What’s the total fidelity?

11.12.6. Exercise 6: Error Analysis

What happens if corrections are applied incorrectly?

- Apply X when you should apply Z
- Skip a required correction
- Apply an unnecessary correction

Measure the fidelity in each case.

11.12.7. Exercise 7: Custom Protocol

Design a teleportation variant that uses $|\Psi^+\rangle = (|01\rangle + |10\rangle)/\sqrt{2}$ instead of $|\Phi^+\rangle$.

- How do the corrections change?
- Implement and verify it works.

11.13. Next Steps

You've mastered adaptive corrections! Next, you'll learn about:

- **Pattern execution** - Simulating full MBQC patterns
- **Pattern validation** - Cross-checking with circuit simulation
- **Photonic backends** - Real hardware compilation

Next: Chapter 12: Pattern Execution and Validation (coming soon)

See also:

- [Chapter 10: Measurement Pattern Generation](#) - Pattern basics
- [Chapter 6: Measurement Patterns Theory](#) - Theoretical background

Part IV.

Part IV: Causal Structure as a Programming Primitive

12. Circuit Compilation for MBQC

In the previous chapters, you learned how to generate individual gate patterns and combine them. Now you'll discover how to compile **complete quantum circuits** into MBQC measurement patterns - the bridge between familiar circuit diagrams and the cluster state model.

12.1. From Circuits to Clusters

Quantum circuits and MBQC are two different computational models that achieve the same goal. Circuit compilation is the process of translating a sequence of gates into an equivalent measurement pattern.

12.1.1. The Circuit Model vs MBQC

Aspect	Circuit Model	MBQC
Computation	Apply gates sequentially	Measure prepared cluster
Resources	Qubits reused	Fresh qubits for each operation
Parallelism	Limited by gate dependencies	Natural from graph structure

12. Circuit Compilation for MBQC

Aspect	Circuit Model	MBQC
Error model	Gate errors	Measurement errors

Key insight: Any quantum circuit can be compiled to MBQC, but the resulting pattern may use more qubits.

12.2. Building a Circuit Compiler

Let's build a simple circuit compiler step by step.

12.2.1. Step 1: Define a Circuit

A circuit is a sequence of gates applied to qubits:

```
from dataclasses import dataclass
from typing import List, Tuple

@dataclass
class Gate:
    """Represents a quantum gate in a circuit."""
    name: str          # Gate name: "H", "X", "CNOT", etc.
    qubits: Tuple[int, ...] # Qubit indices
    params: Tuple[float, ...] = () # Optional parameters (angles)

class QuantumCircuit:
    """Simple quantum circuit representation."""

    def __init__(self, num_qubits: int):
        self.num_qubits = num_qubits
```

12.2. Building a Circuit Compiler

```
self.gates: List[Gate] = []

def h(self, qubit: int):
    """Add Hadamard gate."""
    self.gates.append(Gate("H", (qubit,)))
    return self

def x(self, qubit: int):
    """Add Pauli-X gate."""
    self.gates.append(Gate("X", (qubit,)))
    return self

def z(self, qubit: int):
    """Add Pauli-Z gate."""
    self.gates.append(Gate("Z", (qubit,)))
    return self

def cnot(self, control: int, target: int):
    """Add CNOT gate."""
    self.gates.append(Gate("CNOT", (control, target)))
    return self

def cz(self, qubit1: int, qubit2: int):
    """Add CZ gate."""
    self.gates.append(Gate("CZ", (qubit1, qubit2)))
    return self

def rx(self, qubit: int, angle: float):
    """Add X rotation."""
    self.gates.append(Gate("RX", (qubit,), (angle,)))
    return self

def ry(self, qubit: int, angle: float):
```

12. Circuit Compilation for MBQC

```
        """Add Y rotation."""
        self.gates.append(Gate("RY", (qubit,), (angle,)))
        return self

    def rz(self, qubit: int, angle: float):
        """Add Z rotation."""
        self.gates.append(Gate("RZ", (qubit,), (angle,)))
        return self
```

12.2.2. Step 2: The Compilation Function

```
from qrl.mbcc import (
    generate_single_qubit_gate_pattern,
    generate_cnot_pattern,
    generate_cz_pattern,
    generate_rotation_pattern,
    combine_patterns,
    MeasurementPattern
)

def compile_circuit(circuit: QuantumCircuit) -> MeasurementPattern:
    """
    Compile a quantum circuit to an MBQC measurement pattern.

    Args:
        circuit: QuantumCircuit to compile

    Returns:
        MeasurementPattern implementing the circuit
    """
    if not circuit.gates:
```

12.2. Building a Circuit Compiler

```
# Empty circuit - just prepare qubits
return MeasurementPattern(
    preparation=list(range(circuit.num_qubits)),
    entanglement=[],
    measurements=[],
    corrections=[],
    output_qubits=list(range(circuit.num_qubits)),
    description="Empty circuit"
)

# Compile each gate to a pattern
patterns = []
for gate in circuit.gates:
    pattern = compile_gate(gate)
    patterns.append(pattern)

# Combine all patterns sequentially
result = patterns[0]
for pattern in patterns[1:]:
    result = combine_patterns(result, pattern)

result.description = f"Compiled circuit ({len(circuit.gates)} gates)"
return result

def compile_gate(gate: Gate) -> MeasurementPattern:
    """Compile a single gate to a measurement pattern."""

    if gate.name == "H":
        return generate_single_qubit_gate_pattern("H", gate.qubits[0])

    elif gate.name == "X":
        return generate_single_qubit_gate_pattern("X", gate.qubits[0])
```

12. Circuit Compilation for MBQC

```
elif gate.name == "Z":
    return generate_single_qubit_gate_pattern("Z", gate.qubits[0])

elif gate.name == "S":
    return generate_single_qubit_gate_pattern("S", gate.qubits[0])

elif gate.name == "T":
    return generate_single_qubit_gate_pattern("T", gate.qubits[0])

elif gate.name == "CNOT":
    return generate_cnot_pattern(gate.qubits[0], gate.qubits[1])

elif gate.name == "CZ":
    return generate_cz_pattern(gate.qubits[0], gate.qubits[1])

elif gate.name == "RX":
    return generate_rotation_pattern("X", gate.params[0], gate.qubits[0])

elif gate.name == "RY":
    return generate_rotation_pattern("Y", gate.params[0], gate.qubits[0])

elif gate.name == "RZ":
    return generate_rotation_pattern("Z", gate.params[0], gate.qubits[0])

else:
    raise ValueError(f"Unknown gate: {gate.name}")
```

12.3. Tutorial: Compile a Bell State Circuit

The standard circuit for creating a Bell state is:

q0: 

12.3. Tutorial: Compile a Bell State Circuit

q1: ———X—
 |

Let's compile it:

```
# Create the circuit
circuit = QuantumCircuit(2)
circuit.h(0)      # Hadamard on qubit 0
circuit.cnot(0, 1) # CNOT with control=0, target=1

print(f"Circuit: {len(circuit.gates)} gates")
for gate in circuit.gates:
    print(f" {gate.name} on qubits {gate.qubits}")

# Compile to MBQC pattern
pattern = compile_circuit(circuit)

print(f"\nCompiled pattern:")
print(pattern)
```

Output:

```
Circuit: 2 gates
  H on qubits (0,)
  CNOT on qubits (0, 1)

Compiled pattern:
Pattern: Compiled circuit (2 gates)
Qubits: 6
Preparation: 6 qubits in |+)
Entanglement: 4 CZ gates
Measurements: 3
Corrections: 4
```

12. Circuit Compilation for MBQC

Output qubits: [0, 5]
Measurement depth: 3

12.3.1. Resource Analysis

Compare the compiled pattern with the direct Bell state pattern:

```
from qrl.mbqc import generate_bell_state_pattern

direct_pattern = generate_bell_state_pattern()
compiled_pattern = compile_circuit(circuit)

print("Resource Comparison:")
print(f"{'Resource':<20} {'Direct':>10} {'Compiled':>10}")
print("-" * 42)
print(f"{'Qubits':<20} {direct_pattern.num_qubits:>10} {compiled_pattern.num_qubits:>10}")
print(f"{'CZ gates':<20} {len(direct_pattern.entanglement):>10} {len(compiled_pattern.entanglement):>10}")
print(f"{'Measurements':<20} {len(direct_pattern.measurements):>10} {len(compiled_pattern.measurements):>10}")
print(f"{'Depth':<20} {direct_pattern.measurement_depth:>10} {compiled_pattern.measurement_depth:>10}")
```

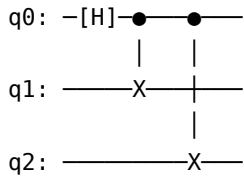
Output:

```
Resource Comparison:
Resource                Direct    Compiled
-----
Qubits                  2         6
CZ gates                 1         4
Measurements            0         3
Depth                   0         3
```

Key insight: The direct pattern is much more efficient! This is because it exploits the structure of the Bell state directly, while the compiled version naively translates each gate.

12.4. Tutorial: GHZ State via Circuit

The GHZ state circuit extends the Bell pattern:



```
# GHZ circuit
ghz_circuit = QuantumCircuit(3)
ghz_circuit.h(0)
ghz_circuit.cnot(0, 1)
ghz_circuit.cnot(0, 2)

ghz_pattern = compile_circuit(ghz_circuit)

print("GHZ Circuit Compilation:")
print(f"  Gates: {len(ghz_circuit.gates)}")
print(f"  Compiled qubits: {ghz_pattern.num_qubits}")
print(f"  Compiled measurements: {len(ghz_pattern.measurements)}")
print(f"  Compiled depth: {ghz_pattern.measurement_depth}")

# Compare with direct
from qrl.mbc import generate_ghz_state_pattern
direct_ghz = generate_ghz_state_pattern(3)

print(f"\nDirect GHZ pattern:")
print(f"  Qubits: {direct_ghz.num_qubits}")
print(f"  Measurements: {len(direct_ghz.measurements)}")
print(f"  Depth: {direct_ghz.measurement_depth}")
```

12. Circuit Compilation for MBQC

Output:

GHZ Circuit Compilation:

Gates: 3

Compiled qubits: 10

Compiled measurements: 5

Compiled depth: 5

Direct GHZ pattern:

Qubits: 3

Measurements: 0

Depth: 0

12.5. The Compilation-Optimization Tradeoff

Why is there such a big difference between compiled and direct patterns?

12.5.1. Naive Compilation

Naive compilation treats each gate independently:

- H gate: 2 qubits, 1 measurement
- CNOT gate: 4 qubits, 2 measurements
- Total: 6 qubits, 3 measurements

12.5.2. Direct Pattern

The direct Bell state pattern recognizes that:

- Bell state = CZ on two $|+\rangle$ qubits
- No measurements needed!
- Total: 2 qubits, 0 measurements

12.5.3. When to Use Each Approach

Approach	Use When
Direct patterns	Known quantum states (Bell, GHZ, W)
Compiled circuits	General algorithms, flexibility needed
Optimized compilation	Production systems (advanced)

12.6. Tutorial: Quantum Teleportation Circuit

Quantum teleportation is a classic algorithm. Let's compile it:

```
import numpy as np

def teleportation_circuit():
    """
    Quantum teleportation circuit.

    Qubits:
    - q0: State to teleport ( $|\psi\rangle$ )
    - q1: Alice's half of Bell pair
    - q2: Bob's half of Bell pair (receives  $|\psi\rangle$ )
    """
    circuit = QuantumCircuit(3)

    # Step 1: Create Bell pair between q1 and q2
    circuit.h(1)
    circuit.cnot(1, 2)

    # Step 2: Bell measurement on q0 and q1
    circuit.cnot(0, 1)
    circuit.h(0)
```

12. Circuit Compilation for MBQC

```
# Note: Classical measurement and corrections would follow
# In MBQC, corrections are built into the pattern

return circuit

teleport_circuit = teleportation_circuit()
teleport_pattern = compile_circuit(teleport_circuit)

print("Teleportation Circuit:")
print(f"  Gates: {len(teleport_circuit.gates)}")
for g in teleport_circuit.gates:
    print(f"    {g.name} on {g.qubits}")

print(f"\nCompiled Pattern:")
print(f"  Qubits: {teleport_pattern.num_qubits}")
print(f"  Measurements: {len(teleport_pattern.measurements)}")
print(f"  Depth: {teleport_pattern.measurement_depth}")
```

12.7. Circuit Depth Analysis

Understanding circuit depth helps optimize compilation:

```
def analyze_circuit_depth(circuit: QuantumCircuit) -> dict:
    """
    Analyze the depth of a quantum circuit.

    Returns dict with depth info per qubit.
    """
    # Track when each qubit is last used
    qubit_depth = {i: 0 for i in range(circuit.num_qubits)}
```

12.7. Circuit Depth Analysis

```
for gate in circuit.gates:
    # Current depth is max of all involved qubits
    current_depth = max(qubit_depth[q] for q in gate.qubits)

    # After this gate, all involved qubits have depth + 1
    new_depth = current_depth + 1
    for q in gate.qubits:
        qubit_depth[q] = new_depth

return {
    'per_qubit': qubit_depth,
    'total': max(qubit_depth.values()),
    'gates': len(circuit.gates)
}

# Analyze our circuits
bell_analysis = analyze_circuit_depth(circuit)
ghz_analysis = analyze_circuit_depth(ghz_circuit)

print("Bell Circuit Depth Analysis:")
print(f"  Circuit depth: {bell_analysis['total']}")
print(f"  Per qubit: {bell_analysis['per_qubit']}")

print("\nGHZ Circuit Depth Analysis:")
print(f"  Circuit depth: {ghz_analysis['total']}")
print(f"  Per qubit: {ghz_analysis['per_qubit']}")
```

Output:

```
Bell Circuit Depth Analysis:
  Circuit depth: 2
  Per qubit: {0: 2, 1: 2}
```

12. Circuit Compilation for MBQC

GHZ Circuit Depth Analysis:

Circuit depth: 3

Per qubit: {0: 3, 1: 2, 2: 3}

12.8. Resource Estimation

Before compiling, estimate the resources needed:

```
def estimate_resources(circuit: QuantumCircuit) -> dict:
    """
    Estimate MBQC resources for a circuit without compiling.

    This is faster than full compilation for resource planning.
    """
    # Resource costs per gate type
    costs = {
        'H': {'qubits': 2, 'measurements': 1, 'cz_gates': 1},
        'X': {'qubits': 2, 'measurements': 1, 'cz_gates': 1},
        'Z': {'qubits': 2, 'measurements': 1, 'cz_gates': 1},
        'S': {'qubits': 2, 'measurements': 1, 'cz_gates': 1},
        'T': {'qubits': 2, 'measurements': 1, 'cz_gates': 1},
        'RX': {'qubits': 2, 'measurements': 1, 'cz_gates': 1},
        'RY': {'qubits': 2, 'measurements': 1, 'cz_gates': 1},
        'RZ': {'qubits': 2, 'measurements': 1, 'cz_gates': 1},
        'CNOT': {'qubits': 4, 'measurements': 2, 'cz_gates': 3},
        'CZ': {'qubits': 2, 'measurements': 0, 'cz_gates': 1},
    }

    total = {'qubits': 0, 'measurements': 0, 'cz_gates': 0}

    for gate in circuit.gates:
        cost = costs.get(gate.name, {'qubits': 2, 'measurements': 1, 'cz_gates': 1})
```


12.9. Advanced: Parallel Gate Execution

```
        for key in total:
            total[key] += cost[key]

    return total

# Estimate resources
print("Resource Estimates (without compilation):")
print(f"\nBell circuit:")
est = estimate_resources(circuit)
print(f"  Qubits: ~{est['qubits']}, Measurements: ~{est['measurements']}")

print(f"\nGHZ circuit:")
est = estimate_resources(ghz_circuit)
print(f"  Qubits: ~{est['qubits']}, Measurements: ~{est['measurements']}")
```

12.9. Advanced: Parallel Gate Execution

MBQC naturally supports parallelism. Gates on different qubits can be measured simultaneously:

```
def identify_parallel_gates(circuit: QuantumCircuit) -> List[List[Gate]]:
    """
    Group gates into parallel layers.

    Gates in the same layer can be executed simultaneously.
    """
    if not circuit.gates:
        return []

    layers = []
    qubit_layer = {i: -1 for i in range(circuit.num_qubits)}
```

12. Circuit Compilation for MBQC

```
for gate in circuit.gates:
    # Find the earliest layer this gate can go in
    min_layer = max(qubit_layer[q] for q in gate.qubits) + 1

    # Ensure we have enough layers
    while len(layers) <= min_layer:
        layers.append([])

    # Add gate to layer
    layers[min_layer].append(gate)

    # Update qubit layers
    for q in gate.qubits:
        qubit_layer[q] = min_layer

return layers

# Analyze parallelism
parallel_circuit = QuantumCircuit(4)
parallel_circuit.h(0)
parallel_circuit.h(1) # Can run parallel with H(0)
parallel_circuit.h(2) # Can run parallel with H(0), H(1)
parallel_circuit.cnot(0, 1)
parallel_circuit.cnot(2, 3) # Can run parallel with CNOT(0,1)

layers = identify_parallel_gates(parallel_circuit)

print("Parallel Gate Analysis:")
for i, layer in enumerate(layers):
    gates_str = ", ".join(f"{g.name}{g.qubits}" for g in layer)
    print(f"  Layer {i}: {gates_str}")

print(f"\nCircuit depth: {len(layers)} (with parallelism)")
```

12.10. Compilation Strategies

```
print(f"Sequential depth: {len(parallel_circuit.gates)} (without parallelism)")
```

Output:

Parallel Gate Analysis:

Layer 0: H(0,), H(1,), H(2,)

Layer 1: CNOT(0, 1), CNOT(2, 3)

Circuit depth: 2 (with parallelism)

Sequential depth: 5 (without parallelism)

12.10. Compilation Strategies

Different strategies trade off resources vs. complexity:

12.10.1. Strategy 1: Gate-by-Gate (Current)

```
# Simple but uses many qubits
for gate in circuit.gates:
    pattern = combine_patterns(pattern, compile_gate(gate))
```

Pros: Simple, always works **Cons:** High qubit count, no optimization

12.10.2. Strategy 2: Gate Fusion (Advanced)

Combine adjacent single-qubit gates into one rotation:

12. Circuit Compilation for MBQC

```
# H followed by S = specific rotation  
# Can be compiled to single measurement
```

Pros: Fewer measurements **Cons:** Complex to implement

12.10.3. Strategy 3: Pattern Rewriting (Advanced)

Replace patterns with equivalent but more efficient ones:

```
# H-CNOT-H on target = CZ (which is native!)
```

Pros: Significant resource reduction **Cons:** Requires pattern library, verification

12.11. What You Learned

You now understand:

- **Circuit representation** - Gates as data structures
- **Compilation workflow** - Circuit to pattern translation
- **Resource analysis** - Qubits, measurements, depth
- **Efficiency tradeoffs** - Direct vs compiled patterns
- **Parallelism** - Identifying parallel gate opportunities
- **Compilation strategies** - Gate-by-gate vs optimized

12.12. Key Takeaways

1. **Any circuit compiles to MBQC** but efficiency varies
2. **Direct patterns beat naive compilation** for known states

3. **Resource estimation** helps planning before compilation
4. **Parallelism is natural** in MBQC - exploit it!
5. **Optimization is an open research area** - many strategies exist

12.13. Exercises

12.13.1. Exercise 1: Compile a Swap Gate

The SWAP gate exchanges two qubits. It can be built from 3 CNOTs:

$\text{SWAP} = \text{CNOT}(0,1) - \text{CNOT}(1,0) - \text{CNOT}(0,1)$

1. Create a circuit implementing SWAP
2. Compile it to an MBQC pattern
3. How many qubits does it need?
4. What's the measurement depth?

12.13.2. Exercise 2: Resource Comparison

Create circuits for:

- 4-qubit GHZ state (H + 3 CNOTs)
- 5-qubit GHZ state (H + 4 CNOTs)

Compare compiled resources with `generate_ghz_state_pattern(n)`.

12. Circuit Compilation for MBQC

12.13.3. Exercise 3: Parallel Circuit

Design a circuit with maximum parallelism:

- 4 qubits
- Apply H to all qubits (parallel)
- Apply CZ between pairs (0,1) and (2,3) (parallel)
- Apply CNOT(1,2) (sequential)

What's the theoretical minimum depth?

12.13.4. Exercise 4: Rotation Circuit

Create a circuit that applies:

- $R_X(\pi/4)$ to qubit 0
- $R_Y(\pi/3)$ to qubit 1
- $R_Z(\pi/6)$ to qubit 2

All in parallel. Compile and verify the measurement angles match the rotations.

12.13.5. Exercise 5: Teleportation Analysis

Using the teleportation circuit:

1. Compile it to a pattern
2. Count the total resources
3. Compare with the direct teleportation pattern from Chapter 11
4. Identify which gates could be optimized away

12.13.6. Exercise 6: Gate Fusion

Mathematically, $H \cdot S \cdot H = R_X(\pi/2)$ (up to global phase).

1. Create a circuit with H-S-H
2. Compile it (3 measurements)
3. Create a circuit with just $R_X(\pi/2)$
4. Compile it (1 measurement)
5. Verify both produce equivalent patterns

12.13.7. Exercise 7: Depth Optimizer

Write a function that reorders gates to minimize depth:

```
def optimize_depth(circuit: QuantumCircuit) -> QuantumCircuit:
    """Reorder gates to minimize circuit depth."""
    # Your implementation here
    pass
```

Test it on a circuit where gates can be reordered.

12.13.8. Exercise 8: Resource Predictor

Extend `estimate_resources` to also predict:

- Measurement depth (longest chain)
- Memory usage (peak qubits alive)
- Classical communication (for corrections)

12.14. Next Steps

You've learned circuit compilation! Next chapters will cover:

- **Chapter 13:** QRL language syntax and semantics
- **Chapter 14:** Pattern execution and simulation
- **Chapter 15:** Photonic backends and real hardware

Next: Chapter 13: QRL Language Syntax (coming soon)

See also:

- [Chapter 10: Pattern Generation](#) - Individual gate patterns
- [Chapter 11: Adaptive Corrections](#) - Correction mechanics

13. The Causal Type Hierarchy

Everything you have built so far — Bell states, GHZ states, cluster states, measurement patterns, compiled circuits — shares one silent assumption: **the order of operations is fixed before the program runs**. A measurement pattern executes round by round. A circuit applies gates left to right. Even adaptive corrections (Chapter 11), which depend on earlier outcomes, respect a definite order: the correction always comes *after* the measurement that determines it.

This chapter asks the question that Part IV is built around: *what if the causal order itself could be quantum?*

The answer is not science fiction. The **quantum switch** — a process in which the order of two operations is controlled by a qubit in superposition — has been realised experimentally many times since 2015. It is a genuine physical resource: it gives measurable advantages in channel discrimination, communication complexity, and thermodynamics. And yet:

No existing quantum programming language can give the quantum switch a type.

Not because the languages are badly designed — because they are designed around a fixed causal skeleton. A circuit language *is* a data structure for definite causal order. To type the switch, causal structure itself has to become a first-class citizen of the type system.

QRL does this with three types: $\text{PM}(n,d)$, $\text{Switch}(d)$, and $\text{CausalDAG}(n)$. This chapter explains what they mean, the mathematics underneath them (process matrices), and the key distinction the types encode: **causal**

13. The Causal Type Hierarchy

separability versus causal nonseparability. Chapter 14 then plays the game that makes the distinction operational, and Chapter 15 proves the theorem that ties it all together.

13.1. The Problem: A Program With No Type

Here is the quantum switch as pseudocode:

```
switch(A, B, control):  
  if control is  $|0\rangle$ : apply A, then B  
  if control is  $|1\rangle$ : apply B, then A  
  if control is  $|+\rangle$ : ... both orders, coherently
```

The first two branches are ordinary programs. The third is not. When the control qubit is in superposition, the process applies A before B **and** B before A — not probabilistically (that would just be a coin flip between two ordinary programs) but *coherently*, in a way that can later interfere.

Try to write this in a circuit language. A circuit is a list of gates; a list has an order. You can simulate the switch’s *statistics* on a circuit machine (using an extra target copy and controlled-SWAPs), but the simulation is a different process — it uses more queries to A and B than the switch does, which is precisely why the switch gives query-complexity advantages. The object itself — “a process with no definite causal order between A and B ” — has no representation, and therefore no type.

What would it even mean to type it? A type is a static guarantee. The type we want is:

Switch(d) — the type of processes on a d -dimensional target that are causally nonseparable.

To make that precise we need the mathematical object that represents “a process without assuming a causal order”. That object is the **process matrix**.

13.2. Process Matrices: Quantum Mechanics Without Global Order

The process matrix framework (Oreshkov, Costa & Brukner, 2012 — hereafter OCB) starts from a deliberately minimal set of assumptions:

1. There are n **parties** (think: local laboratories), each of which receives a quantum system, acts on it with ordinary quantum mechanics, and sends a system out.
2. **Locally**, quantum mechanics is exactly valid in each lab.
3. **Globally** — and this is the departure — *no causal order between the labs is assumed*.

Each party k has an input Hilbert space $\mathcal{H}_{k_I}$ (the system entering the lab) and an output space $\mathcal{H}_{k_O}$ (the system leaving). Everything outside the labs — the wiring, the background causal structure, whatever routes systems between parties — is described by a single operator:

$$W \in \mathcal{L}(\mathcal{H}_{1_I} \otimes \mathcal{H}_{1_O} \otimes \cdots \otimes \mathcal{H}_{n_I} \otimes \mathcal{H}_{n_O})$$

called the **process matrix**. If party k performs an operation with Choi–Jamiołkowski representation M_k (a completely positive map from k_I to k_O), the probability of the joint outcome is given by a generalised Born rule:

$$P(M_1, \dots, M_n) = \text{Tr}[W (M_1 \otimes \cdots \otimes M_n)].$$

13. The Causal Type Hierarchy

For these numbers to be valid probabilities for *every* choice of local operations, W must satisfy:

Condition	Statement	Meaning
Positivity	$W \geq 0$	probabilities are non-negative
Normalisation	$\text{Tr}[W] = \prod_k d_{k_O}$	probabilities sum to 1
No self-signalling	$\text{Tr}[W(M_1 \otimes \dots \otimes M_n)] \in [0, 1]$ for all CPTP instruments	no party signals to its own past

The first two are cheap linear-algebra checks. The third is the deep one — it is what rules out grandfather paradoxes while still permitting indefinite order — and verifying it in general requires characterising the allowed subspace of operators (an SDP in practice).

Why is this the right generalisation? Because it contains everything you already know as special cases:

- **A quantum state** is a process matrix for parties with trivial output spaces: nothing leaves the labs, the parties just measure. $W = \rho$ and the Born rule reduces to $\text{Tr}[\rho M]$.
- **A quantum channel** from Alice to Bob is a process matrix where Alice's output is wired into Bob's input. Definite order $A \prec B$, built in.
- **A circuit** is a chain of such wirings — definite order all the way down.

The framework's payoff is that it also contains process matrices that are *none of these* — valid W operators compatible with local quantum mechanics that do not correspond to any fixed wiring diagram. The quantum switch is the canonical example.

13.2.1. Two processes you can write down

The simplest valid process matrix is the **identity (no-signalling) process** — the parties share nothing:

$$W_{\text{id}} = \frac{1}{d_I^n} I$$

Nobody can signal to anybody. It is the process-matrix analogue of the maximally mixed state.

The canonical *causally ordered* process is the **channel from A to B** (OCB’s equation 1). A’s output is handed to B’s input through an identity channel, represented by the (unnormalised) maximally entangled state $|\Phi^+\rangle = \frac{1}{\sqrt{d}} \sum_i |ii\rangle$ linking A_O to B_I :

$$W_{A \prec B} = \frac{I_{A_I}}{d} \otimes |\Phi^+\rangle\langle\Phi^+|_{A_O B_I} \otimes I_{B_O}$$

A acts first and can signal to B; B cannot signal to A.

13.3. Process Matrices in QRL

QRL represents these objects directly. `ProcessMatrix` carries the operator W , the party names, and the local dimensions — and it can check the OCB validity conditions:

```
from qrl.causal import identity_process, definite_order_process
import numpy as np

# The no-signalling process for two qubit-parties
W_id = identity_process(n_parties=2)
```

13. The Causal Type Hierarchy

```
W_id.is_positive_semidefinite() # True    – condition 1
W_id.is_normalized()           # True    – condition 2:  $\text{Tr}[W] = 2 \times 2 = 4$ 
W_id.is_valid()                # True
np.trace(W_id.W).real          # 4.0
```

The definite-order process is a one-liner, and QRL can verify *structurally* which causal cone it lives in (using the projection characterisation of Araújo et al., which we meet properly below):

```
W_AB = definite_order_process('AB') # A acts first, signals to B

W_AB.is_valid()                    # True
W_AB.is_causally_ordered_AB()     # True  – lies in the  $A < B$  subspace
W_AB.is_causally_ordered_BA()     # False – B cannot signal to A
```

One structural fact worth pausing on: the identity process lies in **both** causal subspaces.

```
W_id.is_causally_ordered_AB() # True
W_id.is_causally_ordered_BA() # True
```

This is not a bug. “Compatible with $A < B$ ” means *B’s marginal statistics don’t depend on A’s operation* — it does not require that A actually signals. A process with no signalling at all is trivially compatible with every order. Causal *order* is a constraint you can satisfy vacuously; that is exactly why the interesting notion is not “ordered” but **separable**, which we define next.

13.4. Causal Separability

Suppose a process is not purely one order or the other, but a classical mixture: with probability q the wiring is $A < B$, with probability $1 - q$

it is $B \prec A$. Before each run, a coin is flipped and one definite order is realised. Whatever such a process can do, it can do with *pre-established* causal facts.

i Definition: causal separability

A two-party process matrix W is **causally separable** if it can be written as

$$W = q W_{A \prec B} + (1 - q) W_{B \prec A}, \quad q \in [0, 1],$$

where $W_{A \prec B}$ and $W_{B \prec A}$ are valid processes compatible with the definite orders $A \prec B$ and $B \prec A$ respectively. A valid process matrix that admits no such decomposition is **causally nonseparable** (CNS).

Causal nonseparability is the precise, order-theoretic analogue of entanglement. Compare:

	Entanglement	Causal nonseparability
Object	state ρ	process W
“Classical” set	separable states $\sum_i p_i \rho_i^A \otimes \rho_i^B$	causally separable processes $qW_{AB} + (1-q)W_{BA}$
Geometry	convex set	convex set
Detection	entanglement witness	causal witness
Certified by	Bell inequality violation	causal inequality / witness violation

The geometric point matters. The causally separable processes form a **convex set**: mixing two separable processes gives a separable process (flip another coin to decide which mixture you’re in). Causal nonseparability is therefore a statement about lying *outside* a convex body — and, exactly as with entanglement, that is something **no single run can reveal**. Any

13. The Causal Type Hierarchy

individual run of any process yields one definite sequence of outcomes, perfectly consistent with *some* causal ordering. Nonseparability only shows up in the statistics, and you detect it the same way you detect entanglement: with a witness.

13.4.1. Causal witnesses

Because the separable set is convex and closed, the hyperplane separation theorem guarantees that for every causally nonseparable W there exists a Hermitian operator Ω — a **causal witness** (Araújo et al., 2015) — with:

$$\text{Tr}[\Omega W'] \geq 0 \quad \text{for every causally separable } W', \quad \text{Tr}[\Omega W] < 0.$$

A negative expectation value is a certificate: this process is incompatible with any classical mixture of causal orders.

QRL implements the witness built from the OCB causal game (the game itself is the whole of Chapter 14; here we only need the operator). Its expectation value on a process with game-winning probability P_{win} is $\text{Tr}[\Omega W] = 3 - 4P_{\text{win}}$, so the separability threshold $\text{Tr}[\Omega W] \geq 0$ is exactly the classical bound $P_{\text{win}} \leq 3/4$:

```
from qrl.causal import causal_nonseparability_witness

Omega = causal_nonseparability_witness() # Hermitian, 16 × 16 for two qubit-parties

W_AB.witness_value(Omega) # 0.5 ≥ 0 (P_win = 0.625)
W_id.witness_value(Omega) # 1.0 ≥ 0 (P_win = 0.5 — can't even use the one-way ch
```

Both non-negative, as they must be — these are causally separable processes. And convex mixtures stay on the safe side of the hyperplane:


```

from qrl.causal import ProcessMatrix

q = 0.7
W_mix = ProcessMatrix(
    W=q * W_AB.W + (1 - q) * W_id.W,
    parties=['A', 'B'],
    input_dims=[2, 2],
    output_dims=[2, 2],
    description="mixture of A→B and the no-signalling process",
)

W_mix.is_valid()           # True
W_mix.is_causally_separable() # True
W_mix.witness_value(Ω)     # 0.65 = 0.7 × 0.5 + 0.3 × 1.0 (linearity)

```

13.4.2. The process that breaks the bound

Now the quantum switch. For *unitary* channels U_A, U_B on a d -dimensional target, the switch is the isometry

$$V = (U_B U_A) \otimes |0\rangle\langle 0|_C + (U_A U_B) \otimes |1\rangle\langle 1|_C$$

acting on target $\otimes$ control. Its process matrix is the (suitably normalised) Choi–Jamiołkowski operator of V — a rank-one, *pure* process:

```

from qrl.causal import QuantumSwitch, cptp_from_unitary
import numpy as np

sw = QuantumSwitch(
    channel_A=cptp_from_unitary(np.eye(2)),
    channel_B=cptp_from_unitary(np.eye(2)),
)

```

13. The Causal Type Hierarchy

```
W_sw = sw.process_matrix()
W_sw.is_valid()           # True  – a perfectly legal process matrix
W_sw.is_causally_separable() # False – but not a mixture of causal orders
sw.causal_inequality_value() # 0.8535533905932737 = (2 + √2)/4 > 3/4
```

The switch wins the OCB game with probability $(2 + \sqrt{2})/4 \approx 0.854$, so its witness value is

$$\text{Tr}[\Omega W_{\text{sw}}] = 3 - 4 \cdot \frac{2 + \sqrt{2}}{4} = 1 - \sqrt{2} \approx -0.414 < 0.$$

Negative. Certificate delivered: no classical mixture of causal orders reproduces this process.

How *far* outside the separable set is it? The **robustness** r^* is the smallest amount of maximally mixed noise that drags W back into the separable set — i.e. the smallest r such that $\frac{W+rW_{\text{id}}}{1+r}$ is causally separable:

```
W_sw.causal_nonseparability_robustness() # 0.4142135623730949 = √2 - 1
```

$r^* = \sqrt{2} - 1$: the switch tolerates about 41% admixed noise before its causal nonseparability is washed out. (That these two numbers — the witness value $1 - \sqrt{2}$ and the robustness $\sqrt{2} - 1$ — are negatives of each other is not a coincidence; Chapter 14 derives both.)

Note the *identity channels* in the construction: $U_A = U_B = I$. The causal nonseparability of the switch has nothing to do with what A and B compute — it is a property of the *wiring*, the process itself. Even “do nothing, twice, in a superposition of orders” is causally nonseparable.

13.5. QRL's Three Causal Types

We now have the full mathematical cast. Here is how QRL turns it into a type system.

$$\tau ::= \dots \mid \text{PM}(n, d) \mid \text{Switch}(d) \mid \text{CausalDAG}(n)$$

13.5.1. $\text{PM}(n, d)$ — the general process

The type of n -party process matrices with local dimension d . Its introduction rule checks the cheap OCB conditions at the type level:

$$\frac{W \in \mathbb{C}^{d^{2n} \times d^{2n}} \quad W \geq 0 \quad \text{Tr}_{\text{out}}(W) = I_{\text{in}}}{\Gamma \vdash \text{pm}(W, \mathbf{p}) : \text{PM}(n, d)} \quad (\text{T-PM})$$

Everything in this chapter so far inhabits $\text{PM}(2, 2)$: the identity process, $W_{A \prec B}$, the mixture, *and* the switch's process matrix. $\text{PM}(n, d)$ is the ambient type — it promises validity, not causal structure.

13.5.2. $\text{Switch}(d)$ — the certified causally nonseparable process

The type of quantum switches on a d -dimensional target. Its introduction rule is where the physics enters the type system:

$$\frac{\Gamma \vdash f : \text{UniProc}(d) \quad \Gamma \vdash g : \text{UniProc}(d) \quad \Gamma \vdash c : \text{Qubit}}{\Gamma \vdash \text{switch}(f, g, c) : \text{Switch}(d)} \quad (\text{T-Switch})$$

The crucial premise is $\text{UniProc}(d)$ — the channels must be **unitary**, not merely CPTP. This is not pedantry. For non-unitary channels, the physical switch decoheres into an *incoherent classical mixture* of the two orders —

13. The Causal Type Hierarchy

which is, by definition, causally separable. The API will happily build that object for you, but it gets type `PM(2,d)`, never `Switch(d)`. Unitarity is exactly the condition under which coherence between the orders survives, and the type rule enforces it statically.

This is what makes `Switch(d)` a **certified** type. The claim — proved as the Soundness Theorem in Chapter 15 — is:

$$\vdash e : \text{Switch}(d) \implies \llbracket e \rrbracket \in \text{CNS}_{2,d}$$

Any term that type-checks as `Switch(d)` denotes a causally nonseparable process matrix. The type checker is issuing a physics certificate.

13.5.3. `CausalDAG(n)` — definite order, made explicit

The third type covers the opposite regime: processes whose causal structure is definite and *known*, given as a directed acyclic graph with quantum channels along the edges (the quantum causal models of Allen et al., 2017):

$$\frac{V \text{ finite} \quad (V, E) \text{ acyclic} \quad \Phi : E \rightarrow \text{CPTP}}{\Gamma \vdash \text{dag}(V, E, \Phi) : \text{CausalDAG}(|V|)} \quad (\text{T-DAG})$$

Why have this at all, if `PM(n,d)` already contains every definite-order process? Because the DAG supports operations that the bare process matrix does not expose — above all **intervention**. `CausalDAG(n)` is the type at which quantum analogues of Pearl’s do-calculus live:

```
from qrl.causal import QuantumCausalDAG, ctp_from_unitary
import numpy as np

# A three-node chain Z → X → Y with identity mechanisms
plus = np.array([[0.5, 0.5], [0.5, 0.5]], dtype=complex) # |+⟩⟨+|
```

13.5. QRL's Three Causal Types

```

dag = QuantumCausalDAG("Z → X → Y chain")
dag.add_node("Z", dim=2, prior=plus)
dag.add_node("X", dim=2)
dag.add_node("Y", dim=2)
dag.add_channel("Z", "X", ctp_from_unitary(np.eye(2)))
dag.add_channel("X", "Y", ctp_from_unitary(np.eye(2)))

dag.observational_state("Y")
# [[0.5, 0.5],      Y inherits |+⟩|+⟩ from Z through the chain
#  [0.5, 0.5]]

dag.interventional_state("Y", {"X": np.array([[1, 0], [0, 0]], dtype=complex)})
# [[1, 0],         do(X = |0⟩⟨0|) cuts the Z → X arrow:
#  [0, 0]]         Y now sees only what we set X to

```

The intervention $\text{do}(X = |0\rangle\langle 0|)$ severs X from its causes and fixes its state; downstream, Y changes accordingly. Observation and intervention disagree — the defining signature of causal (as opposed to merely statistical) structure. In QRL the interventional query has its own typing rule ($\tau\text{-Do}$), and it is only available at type $\text{CausalDAG}(n)$: intervention *requires* a definite causal arrow to cut.

13.5.4. The hierarchy — and a subtlety

Putting the semantic domains side by side:

$$\underbrace{\llbracket \text{Switch}(d) \rrbracket}_{= \text{CNS}_{2,d}} \subsetneq \llbracket \text{PM}(2,d) \rrbracket \supsetneq \underbrace{\text{definite-order processes}}_{\text{incl. } \llbracket \text{CausalDAG} \rrbracket \text{ semantics}}$$

Every switch denotes a process matrix; strictly fewer process matrices are switches. So you might expect subtyping: $\text{Switch}(d) \leq \text{PM}(2,d)$, with coercions in both directions where dimensions allow.

13. The Causal Type Hierarchy

QRL deliberately does **not** do this — and the direction that is missing matters. The inclusion above holds between *semantic domains*. At the *type* level there is no rule that produces a `Switch(d)` from a `PM(n,d)`: the types are not coercible. If they were, you could take any valid process matrix — say, the thoroughly separable $W_{A \prec B}$ — cast it to `Switch(d)`, and the certificate would be worthless. The only way to inhabit `Switch(d)` is through `T-Switch`, and `T-Switch` only accepts ingredients (unitary channels, a control qubit) from which causal nonseparability provably follows.

This is the design pattern to take away from the chapter:

A certified type is one whose introduction rules are the proof. `Switch(d)` doesn't *assert* causal nonseparability; its typing rule makes anything else unconstructible.

The same pattern appears elsewhere in programming languages — think of a `NonEmptyList` that can only be built from a head and a tail, or a `SortedList` whose only constructor is insertion — but here the invariant being certified is not a data-structure property. It is a statement about the causal structure of physical reality, checked by a compiler.

13.6. So What?

Three types, one hierarchy, one design principle. What has it bought us?

1. **The switch has a type.** The object that circuit languages cannot represent is a first-class value, and its defining physical property is a static guarantee rather than a runtime observation.
2. **Causal structure is now part of a program's interface.** A function that demands `Switch(d)` *knows* — at compile time — that its argument is a genuinely indefinite-order resource. A function returning `CausalDAG(n)` advertises that interventions on its result are meaningful. `PM(n,d)` says: valid physics, no causal promises.

3. **The distinctions are checkable.** Everything in this chapter ran against `qrl.causal`: validity conditions, cone membership, witness values, robustness, interventions. The type system’s semantic claims are grounded in executable linear algebra, not just inference rules on paper.

What we have *not* yet done is justify the two magic numbers that appeared: $P_{\text{win}} = (2 + \sqrt{2})/4$ and the classical bound $3/4$. They come from a game — a beautifully simple one — and playing it carefully is the whole of Chapter 14. The Soundness Theorem, which welds this chapter’s types to that chapter’s numbers, is Chapter 15.

13.7. Exercises

Exercise 1 — Validity conditions (warm-up). Construct `identity_process(n_parties=3)` and predict $\text{Tr}[W]$ before you run `is_normalized()`. Then break it: scale `w_id.w` by 2, rebuild the `ProcessMatrix`, and check which of the two validity conditions fails. Which OCB condition (positivity, normalisation, no-self-signalling) does *neither* of these checks cover, and why is it harder?

Exercise 2 — Witness arithmetic. Using the relation $\text{Tr}[\Omega W] = 3 - 4P_{\text{win}}$, verify by hand the three witness values computed in this chapter (1.0 for the identity process, 0.5 for $W_{A \prec B}$, $1 - \sqrt{2}$ for the switch) from their winning probabilities (0.5, 0.625, $(2 + \sqrt{2})/4$). What winning probability corresponds to witness value exactly 0, and what does that process sit on, geometrically?

Exercise 3 — Convexity, numerically. Sweep q from 0 to 1 in steps of 0.1 for the mixture $q W_{A \prec B} + (1 - q) W_{\text{id}}$ and record `witness_value(omega)` at each step. Confirm linearity. Then argue (two or three sentences, no code) why *no* convex mixture of causally separable processes can ever have a negative witness value — which single property of the trace makes this inevitable?

13. The Causal Type Hierarchy

Exercise 4 — Tensor orderings (a real pitfall). `definite_order_process('BA')` returns a matrix that is numerically *identical* to `definite_order_process('AB')` — only the party labels differ (`['B', 'A']` vs `['A', 'B']`). Explain why. Then explain what would go wrong, physically, if you formed the “mixture” $0.5 * W_{AB} + 0.5 * W_{BA}$ and interpreted it as $\frac{1}{2}W_{A \prec B} + \frac{1}{2}W_{B \prec A}$. What permutation of tensor factors would you need to apply to W_{BA} first?

Exercise 5 — Interventions and d-separation (harder). Extend the chain example to a *common-cause* DAG: one root Z with prior $|+\rangle\langle+|$ and two children $X \leftarrow Z \rightarrow Y$ (identity channels). (a) Compare `observational_state("Y")` before and after `interventional_state("Y", {"X": ...})` — why does intervening on X now do *nothing* to Y ? (b) Check your structural explanation with `is_d_separated`. (c) In the chain $Z \rightarrow X \rightarrow Y$, the same intervention changed Y completely. State the general principle: for which graph structures does $\text{do}(X)$ affect Y ?

13.8. Next Steps

You now hold the three causal types and the concept they hinge on — causal (non)separability, witnessed rather than observed. Next:

- **Chapter 14: The OCB Causal Game** — where $3/4$ and $(2 + \sqrt{2})/4$ actually come from. The game, the classical bound and its proof, the switch strategy, and the three-line QRL program that computes it.
- **Chapter 15: QuantumSwitch Soundness** — the theorem $\vdash e : \text{Switch}(d) \Rightarrow \llbracket e \rrbracket \in \text{CNS}$, its proof, and why McBeth, Sabry, Heunen, and Dave — four serious quantum programming languages — cannot state it.

See also:

- [Chapter 2: Entanglement](#) — the state-level analogue of everything here

- [Chapter 12: Circuit Compilation](#) — the definite-order world this chapter generalises

References for this chapter:

- Oreshkov, Costa & Brukner (2012), *Quantum correlations with no causal order*, Nature Communications **3**, 1092 — the process matrix framework.
- Araújo, Branciard, Costa, Feix, Giarmatzi & Brukner (2015), *Witnessing causal nonseparability*, New Journal of Physics **17**, 102001 — causal witnesses, the projection characterisation of causal cones, and robustness.
- Allen, Barrett, Horsman, Lee & Spekkens (2017), *Quantum common causes and quantum causal models*, Physical Review X **7**, 031021 — the quantum causal models behind CausalDAG(n).

14. The OCB Causal Game

Chapter 13 leaned on two numbers without earning them: the classical bound $P_{\text{win}} \leq 3/4$, and the quantum switch’s score $P_{\text{win}} = (2 + \sqrt{2})/4 \approx 0.854$. They are the numerical content of the whole causal type hierarchy — the gap between them is *why* `Switch(d)` deserves to be its own type. This chapter derives both.

They come from a game. Oreshkov, Costa & Brukner introduced it in the same 2012 paper that gave us process matrices, and it plays the role for causal structure that the CHSH game plays for entanglement: a task with a sharp classical ceiling that a quantum resource provably beats.

14.1. The Rules

Two laboratories, A and B . In each round:

1. A referee generates two independent uniform bits, x and y . Bit x is delivered to lab A , bit y to lab B . Neither lab sees the other’s bit.
2. Each lab performs one local quantum operation — an instrument chosen in advance as a function of its own input bit — and reports a single output bit: a from A , b from B .
3. The round is **won** if

$$a \oplus b = x \wedge y.$$

14. The OCB Causal Game

So for three of the four input pairs ($x \wedge y = 0$) the labs must produce **equal** bits; for $x = y = 1$ they must produce **different** bits. The score is the win probability averaged over the four equally likely inputs:

$$P_{\text{win}} = \frac{1}{4} \sum_{x,y \in \{0,1\}} P(a \oplus b = x \wedge y \mid x, y).$$

Here is the twist that makes it a *causal* game rather than a nonlocality game. The two labs are not simply spacelike-separated with a shared state. They are embedded in a **process** — some background arrangement of wires, channels, and systems that routes quantum information between them. The labs agree in advance on their instruments and on that process. What they are *not* given is a definite causal order between A and B : the process is allowed to be any valid process matrix.

The question is exactly the question of Chapter 13, now made operational: **does having a definite causal order (or a classical mixture of orders) limit what the labs can score?**

14.2. Why Definite Order Caps the Score at 3/4

Fix a definite causal order, say $A \prec B$: information may flow from A 's lab to B 's, never the reverse. (The $B \prec A$ case is identical by symmetry, and Chapter 13 established that a classical mixture of the two is a convex combination, so whatever bound holds for each order holds for the mixture.)

The intuition is a see-saw. In the order $A \prec B$, lab B is *downstream*: the system it operates on has already passed through A , so B 's statistics can be correlated with what A did. Lab A is *upstream*: nothing it touches has been near B , so A 's output cannot reflect B 's input y in any way. Whatever coordination the win condition demands, at most one of the two labs can be adapting to the other; the other is working blind.

14.2. Why Definite Order Caps the Score at 3/4

Make this precise. Under $A \prec B$, lab B cannot signal to lab A , so A 's output marginal is independent of y :

$$P(a \mid x, y) = P(a \mid x).$$

Now look at the win condition column by column:

(x, y)	$x \wedge y$	required relation
$(0, 0)$	0	$a = b$
$(0, 1)$	0	$a = b$
$(1, 0)$	0	$a = b$
$(1, 1)$	1	$a \neq b$

The three “ $a = b$ ” rows want the labs *correlated the same way*; the last row wants them *anti-correlated*. Consider the two rows with $x = 1$: $(1, 0)$ needs $a = b$ and $(1, 1)$ needs $a \neq b$, but A 's bit a has the **same** distribution in both rows (it depends only on $x = 1$, not on y). So the burden of switching from “agree” to “disagree” falls entirely on B — and B must do it *knowing* y but *not* x . When B sees $y = 1$ it does not know whether it is in row $(0, 1)$ (agree) or row $(1, 1)$ (disagree). Any fixed response to $y = 1$ therefore gets one of those two rows wrong.

Carrying the bookkeeping through all four rows — a short calculation, or OCB's causal inequality applied to the cone characterisation from Chapter 13 — gives

$$\boxed{P_{\text{win}}^{\text{causally separable}} \leq \frac{3}{4}.$$

The bound is tight: with order $A \prec B$, let A output $a = 0$ always and let B output $b = 0$ when $y = 0$ and a fair coin when $y = 1$. Rows $(0, 0)$, $(1, 0)$ win outright; rows $(0, 1)$, $(1, 1)$ win with probability $1/2$ each. Average: $\frac{1}{4}(1 + 1 + \frac{1}{2} + \frac{1}{2}) = \frac{3}{4}$.

14. The OCB Causal Game

14.2.1. Seeing the bound in QRL

Chapter 13 met the causally ordered processes as `ProcessMatrix` values. Their game scores are computed directly by `causal_inequality_value()`:

```
from qrl.causal import identity_process, definite_order_process

identity_process(n_parties=2).causal_inequality_value()    # 0.5    - no signalling at
definite_order_process('AB').causal_inequality_value()    # 0.625 - one-way channel A
```

The no-signalling process cannot even use the “downstream” adaptation — B learns nothing about A — and scores a flat $1/2$, no better than answering at random. The one-way channel $A \prec B$ does better, 0.625 , by letting B adapt to A ; but it is still short of $3/4$ because these specific processes are not using the optimal instruments. What matters is that **every** causally separable process, with every choice of instruments, stays at or below $3/4$. That ceiling is the whole point.

14.3. The Switch Strategy

Now embed the two labs in a **quantum switch** instead of a definite wiring. Recall its isometry (Chapter 13), for unitary operations U_A, U_B on the target and a control qubit C :

$$V = (U_B U_A) \otimes |0\rangle\langle 0|_C + (U_A U_B) \otimes |1\rangle\langle 1|_C.$$

Set the control to $|+\rangle$. The process is now a *coherent superposition* of “ A then B ” and “ B then A ”. Neither lab is downstream; neither is upstream; the amplitudes for the two orders are both present and can interfere.

The labs’ strategy is the one already coded into QRL’s score operator:

14.4. Computing It in QRL

- **On input** 0, measure the target in the Z basis and report the outcome.
- **On input** 1, measure in the X basis and report the outcome.

Why this beats $3/4$: the two orders differ by whether A 's operation or B 's comes last, and Z and X measurements *do not commute*, so the order genuinely changes the target the other lab sees. With the control in $|+\rangle$, the win amplitude picks up a contribution from each order, and — exactly as in the CHSH derivation — the two contributions add constructively along a direction 45° between the axes. The interference term carries a factor $\cos(\pi/4) = 1/\sqrt{2}$, and the score becomes

$$P_{\text{win}} = \frac{1}{2} + \frac{1}{2\sqrt{2}} = \frac{2 + \sqrt{2}}{4} \approx 0.854.$$

The same $\sqrt{2}$ that sets Tsirelson's bound for CHSH sets the causal advantage here. It is not a coincidence: both are the operator norm of a sum of two anticommuting ± 1 observables.

14.4. Computing It in QRL

Chapter 13 asserted the number; the surface language computes it. Three lines:

Listing 14.1 ocb.qrl

```
let A : UniProc(2) = ctp([1, 0], [0, 1]) in  -- identity unitary
let B : UniProc(2) = ctp([1, 0], [0, 1]) in  -- identity unitary
switch(A, B, |>)
```

14. The OCB Causal Game

```
$ qrl exec ocb.qrl
ocb.qrl Process(2x2, P_win=0.8536, robustness=0.4142)
```

The program type-checks as `Switch(2)` — both operations are `UniProc(2)`, so `T-Switch` applies — and the interpreter takes the `E-Switch-Coherent` reduction, builds the switch process matrix, and scores it. Note the operations are *identity* unitaries: as Chapter 13 stressed, the causal advantage is a property of the process, not of what the labs compute.

The same computation through the Python API, which exposes the intermediate objects:

```
from qrl.causal import QuantumSwitch, ctp_from_unitary
import numpy as np

sw = QuantumSwitch(
    channel_A=ctp_from_unitary(np.eye(2)),
    channel_B=ctp_from_unitary(np.eye(2)),
)

sw.causal_inequality_value() # 0.8535533905932737 = (2 + √2)/4
```

14.5. The Score Operator

`causal_inequality_value()` is not a Monte Carlo simulation of the game — it evaluates it in closed form. The game’s entire payoff structure is packed into one Hermitian operator S , the **score operator**, built from the Choi–Jamiołkowski representations of the optimal instruments:

$$S = \sum_{\substack{x,y,a,b \\ a \oplus b = x \wedge y}} J_A^{x,a} \otimes J_B^{y,b},$$

14.5. The Score Operator

where $J^{x,a}$ is the Choi operator of “measure in the x -basis, obtain outcome a ”. Summing over exactly the winning (x, y, a, b) tuples means that for any process matrix W ,

$$P_{\text{win}}(W) = \frac{1}{4} \text{Tr}[W S].$$

This is the generalised Born rule of Chapter 13 with S in place of a product of local operations: the game score is *linear* in the process. That linearity is what turns the $3/4$ bound into a **witness**. Define

$$\Omega = \frac{3}{4} I - S.$$

Then for any valid two-party qubit process, which Chapter 13’s normalisation condition fixes at $\text{Tr}[W] = 4$,

$$\text{Tr}[\Omega W] = \frac{3}{4} \text{Tr}[W] - \text{Tr}[S W] = 3 - 4 P_{\text{win}}(W).$$

The classical ceiling $P_{\text{win}} \leq 3/4$ is exactly $\text{Tr}[\Omega W] \geq 0$. This is the operator Chapter 13 called `causal_nonseparability_witness()`, and its non-negativity on causally separable processes is the OCB causal inequality in its sharpest form.

```
from qrl.causal import causal_nonseparability_witness, identity_process, definite_order_process

Omega = causal_nonseparability_witness()

identity_process(n_parties=2).witness_value(Omega)    # 1.0    = 3 - 4(0.5)
definite_order_process('AB').witness_value(Omega)    # 0.5    = 3 - 4(0.625)
```

Both non-negative — as they must be for causally separable processes. For the switch, the relation $\text{Tr}[\Omega W] = 3 - 4 P_{\text{win}}$ gives

$$\text{Tr}[\Omega W_{\text{sw}}] = 3 - 4 \cdot \frac{2 + \sqrt{2}}{4} = 1 - \sqrt{2} \approx -0.414 < 0.$$

Negative: the certificate of Chapter 13, now with its provenance. (The switch’s process matrix is expressed over target $\otimes$ control rather than

14. The OCB Causal Game

$A_I \otimes A_O \otimes B_I \otimes B_O$, so `witness_value` cannot be applied to it *directly* — but the linear relation is basis-independent, and `causal_inequality_value()` returns the score that feeds it.)

14.6. Robustness: How Far Outside the Bound

A negative witness value says the switch is causally nonseparable. **Robustness** says by how much. It is the smallest fraction of maximally-mixed noise that, blended in, drags the process back onto the classical side:

$$r^*(W) = \min \left\{ r \geq 0 : \frac{W + r W_{\text{id}}}{1 + r} \text{ is causally separable} \right\}.$$

Because the score is linear and W_{id} scores exactly 1/2, the blended process scores

$$P_{\text{win}}\left(\frac{W + r W_{\text{id}}}{1 + r}\right) = \frac{P_{\text{win}}(W) + r/2}{1 + r},$$

and setting this equal to the threshold 3/4 and solving for r gives

$$r^*(W) = 4 \left(P_{\text{win}}(W) - \frac{3}{4} \right)$$

whenever $P_{\text{win}}(W) > 3/4$, and 0 otherwise. For the switch:

$$r^* = 4 \left(\frac{2 + \sqrt{2}}{4} - \frac{3}{4} \right) = \sqrt{2} - 1 \approx 0.414.$$

```
W_sw = sw.process_matrix()
W_sw.causal_nonseparability_robustness() # 0.4142135623730949 = sqrt(2) - 1
```

The switch tolerates roughly 41% admixed noise before its causal nonseparability is gone. And now the coincidence Chapter 13 flagged is explained: the witness value is $3 - 4P_{\text{win}} = -(r^*)$ exactly, whenever the bound is exceeded. The two numbers $1 - \sqrt{2}$ and $\sqrt{2} - 1$ are negatives of each other because they are the same linear function of P_{win} , read with opposite sign.

14.7. So What?

A gate-based language can compute every number in this chapter. Give it the switch isometry V , let it build the Choi operator, contract it with S , and it will print 0.8536. The number is not the point.

The point is the *type*. When `ocb.qrl` type-checks as `Switch(2)`, the compiler has certified — before any number is computed — that this process lies outside the causally separable set. $P_{\text{win}} > 3/4$ is not a measured result to be checked against a threshold; it is a static consequence of the term having the type it has. That is the guarantee Chapter 15 makes precise and proves: the Soundness Theorem, which says the type checker’s certificate is never wrong.

14.8. Exercises

Exercise 1 — The classical bound by hand. Enumerate all deterministic strategies for the order $A \prec B$ in which a depends only on x and b depends on y and on a . There are few enough to list. Show that none exceeds $P_{\text{win}} = 3/4$, and identify one that achieves it. Then explain in one sentence why allowing shared randomness cannot help.

Exercise 2 — Linearity of the score. Using `causal_inequality_value()`, compute P_{win} for the mixtures $qW_{A \prec B} + (1 - q)W_{\text{id}}$ at $q = 0, 0.25, 0.5, 0.75, 1$ (build each with `ProcessMatrix`, as in Chapter 13). Confirm the results lie on the straight line $P_{\text{win}}(q) = \frac{1}{2} + \frac{1}{8}q$. Which property of $\text{Tr}[WS]$ forces this?

Exercise 3 — Robustness from the score. Derive $r^* = 4(P_{\text{win}} - 3/4)$ from scratch: write down the score of the noise-blended process, set it to the threshold $3/4$, solve for r . Then verify numerically that `W_sw.causal_nonseparability_robustness()` agrees with `4(W_sw.causal_inequality_value()$ - 0.75)$`.

14. The OCB Causal Game

Exercise 4 — Why Z and X (harder). The switch strategy measures in the Z basis on input 0 and the X basis on input 1. Replace the X -basis choice with a measurement at angle θ from Z in the Z - X plane, and (using the API to rebuild the score operator, or by hand) find $P_{\text{win}}(\theta)$. Show it is maximised at $\theta = \pi/2$ and that the maximum is $(2 + \sqrt{2})/4$. Where does the $\sqrt{2}$ enter?

Exercise 5 — A game the switch cannot win (open-ended). The switch beats the classical bound for *this* win condition, $a \oplus b = x \wedge y$. Construct a different two-input, two-output win condition for which the quantum switch scores no better than the best causally ordered process. What feature of $a \oplus b = x \wedge y$ — think about which rows demand correlation versus anti-correlation — is the switch exploiting, and how would you design a predicate that removes it?

14.9. Next Steps

You now have the gap: $3/4$ for anything with a definite causal order, $(2 + \sqrt{2})/4$ for the switch, and the robustness $\sqrt{2} - 1$ measuring the distance between them.

- **Chapter 15: QuantumSwitch Soundness** — the theorem that connects this gap to the type system. It states and proves $\vdash e : \text{Switch}(d) \Rightarrow \llbracket e \rrbracket \in \text{CNS}_{2,d}$: every well-typed switch term denotes a process on the far side of the $3/4$ bound. And it explains why McBeth, Sabry, Heunen, and Dave — four serious quantum programming languages — cannot even state it.

See also:

- [Chapter 13: The Causal Type Hierarchy](#) — process matrices, causal separability, and the causal witness used here
- [Chapter 2: Entanglement](#) — the CHSH game, whose Tsirelson bound is the same $\sqrt{2}$

References for this chapter:

- Oreshkov, Costa & Brukner (2012), *Quantum correlations with no causal order*, Nature Communications **3**, 1092 — the causal game and the causal inequality.
- Araújo, Branciard, Costa, Feix, Giarmatzi & Brukner (2015), *Witnessing causal nonseparability*, New Journal of Physics **17**, 102001 — the score operator, the witness, and the robustness measure.
- Chiribella, D’Ariano, Perinotti & Valiron (2013), *Quantum computations without definite causal structure*, Physical Review A **88**, 022318 — the quantum switch as a higher-order operation.

15. Coming Soon

This chapter is under development...

A. Installation

Installation guide coming soon...

B. API Reference

API documentation coming soon...

C. Exercise Solutions

This appendix contains solutions to all exercises in the book.

C.1. Chapter 1: Quantum Mechanics Basics

C.1.1. Solution 1.1: Normalization

Given the unnormalized state $|\psi\rangle = 2|0\rangle + 2i|1\rangle$:

(a) Calculate $\langle\psi|\psi\rangle$:

$$\langle\psi|\psi\rangle = |2|^2 + |2i|^2 = 4 + 4 = 8$$

(b) Normalize the state:

$$|\psi_{\text{norm}}\rangle = \frac{|\psi\rangle}{\sqrt{\langle\psi|\psi\rangle}} = \frac{2|0\rangle + 2i|1\rangle}{\sqrt{8}} = \frac{2|0\rangle + 2i|1\rangle}{2\sqrt{2}} = \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle$$

(c) Verify normalization:

$$\langle\psi_{\text{norm}}|\psi_{\text{norm}}\rangle = \left|\frac{1}{\sqrt{2}}\right|^2 + \left|\frac{i}{\sqrt{2}}\right|^2 = \frac{1}{2} + \frac{1}{2} = 1 \quad \checkmark$$

[← Back to Exercise 1.1](#)

C.1.2. Solution 1.2: Measurement Probabilities

Given $|\psi\rangle = \frac{1}{2}|0\rangle + \frac{\sqrt{3}}{2}|1\rangle$:

(a) Measurement probabilities in Z-basis:

$$P(0) = \left| \frac{1}{2} \right|^2 = \frac{1}{4}$$

$$P(1) = \left| \frac{\sqrt{3}}{2} \right|^2 = \frac{3}{4}$$

(b) Verification:

$$P(0) + P(1) = \frac{1}{4} + \frac{3}{4} = 1 \quad \checkmark$$

(c) Post-measurement state after obtaining outcome 1:

$$|\psi_{\text{post}}\rangle = |1\rangle$$

The state collapses to the measured basis state.

[← Back to Exercise 1.2](#)

C.1.3. Solution 1.3: Basis Transformation

Express $|1\rangle$ in the X-basis.

Using the inverse relation from the hint:

$$|1\rangle = \frac{1}{\sqrt{2}}(|+\rangle - |-\rangle)$$

Verification: Substituting the definitions of $|+\rangle$ and $|-\rangle$:

$$\begin{aligned} \frac{1}{\sqrt{2}}(|+\rangle - |-\rangle) &= \frac{1}{\sqrt{2}} \left[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) - \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \right] \\ &= \frac{1}{2}[(|0\rangle + |1\rangle) - (|0\rangle - |1\rangle)] = \frac{1}{2}(2|1\rangle) = |1\rangle \quad \checkmark \end{aligned}$$

[← Back to Exercise 1.3](#)

C.1.4. Solution 1.4: Cross-Basis Measurement

Given $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$:

(a) Z-basis measurement probabilities:

$$P(0) = \left| \frac{1}{\sqrt{2}} \right|^2 = \frac{1}{2}, \quad P(1) = \left| \frac{1}{\sqrt{2}} \right|^2 = \frac{1}{2}$$

(b) X-basis measurement probabilities:

Since $|+\rangle$ is itself a basis state of the X-basis:

C. Exercise Solutions

$$P(+) = 1, \quad P(-) = 0$$

(c) Explanation:

The state $|+\rangle$ is a **superposition** in the Z-basis, so measurements yield random outcomes (50/50). However, $|+\rangle$ is an **eigenstate** of the X-basis, so X-measurements always yield the same result with certainty.

This illustrates **basis-dependent measurement**: the same quantum state appears “definite” in one basis but “random” in another. The choice of measurement basis determines whether we see quantum randomness or deterministic outcomes.

[← Back to Exercise 1.4](#)

C.1.5. Solution 1.5: Global Phase

Show that $|\psi\rangle = |0\rangle$ and $|\psi'\rangle = -|0\rangle = e^{i\pi}|0\rangle$ are physically equivalent.

Z-basis measurements:

For $|\psi\rangle = |0\rangle$:

$$P(0) = |1|^2 = 1, \quad P(1) = |0|^2 = 0$$

For $|\psi'\rangle = -|0\rangle$:

$$P(0) = |-1|^2 = 1, \quad P(1) = |0|^2 = 0$$

Identical probabilities. ✓

X-basis measurements:

Express both states in X-basis using $|0\rangle = \frac{1}{\sqrt{2}}(|+\rangle + |-\rangle)$:

For $|\psi\rangle = |0\rangle = \frac{1}{\sqrt{2}}|+\rangle + \frac{1}{\sqrt{2}}|-\rangle$:

$$P(+)=\frac{1}{2}, \quad P(-)=\frac{1}{2}$$

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

$$P(+)=\left|-\frac{1}{\sqrt{2}}\right|^2=\frac{1}{2}, \quad P(-)=\left|-\frac{1}{\sqrt{2}}\right|^2=\frac{1}{2}$$

Identical probabilities. ✓

Since both states yield identical measurement probabilities in every basis, they are **physically indistinguishable**. The global phase factor $e^{i\pi} = -1$ has no observable effect.

← Back to Exercise 1.5

C.1.6. Solution 1.6: Bloch Sphere Coordinates

Given $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{e^{i\pi/4}}{\sqrt{2}}|1\rangle$

The general Bloch sphere parameterization is:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$$

Comparing coefficients:

For the $|0\rangle$ coefficient:

$$\cos\frac{\theta}{2} = \frac{1}{\sqrt{2}} \implies \frac{\theta}{2} = \frac{\pi}{4} \implies \theta = \frac{\pi}{2}$$

C. Exercise Solutions

For the $|1\rangle$ coefficient:

$$e^{i\phi} \sin \frac{\theta}{2} = \frac{e^{i\pi/4}}{\sqrt{2}}$$

Since $\sin \frac{\pi}{4} = \frac{1}{\sqrt{2}}$:

$$e^{i\phi} \cdot \frac{1}{\sqrt{2}} = \frac{e^{i\pi/4}}{\sqrt{2}} \implies e^{i\phi} = e^{i\pi/4} \implies \phi = \frac{\pi}{4}$$

Answer: $\theta = \frac{\pi}{2}$ (on the equator), $\phi = \frac{\pi}{4}$ (45° from the $|+\rangle$ state toward $|+i\rangle$)

[← Back to Exercise 1.6](#)

C.2. Chapter 2: Entanglement

C.2.1. Solution 2.1: Identifying Entanglement

(a) $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)$

Factor out the first qubit:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle_A \otimes (|0\rangle + |1\rangle)_B = |0\rangle_A \otimes |+\rangle_B$$

Separable - this is a product state.

(b) $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$

Factor out the second qubit:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)_A \otimes |0\rangle_B = |+\rangle_A \otimes |0\rangle_B$$

Separable - this is a product state.

(c) $|\psi\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$

Factor:

$$|\psi\rangle = \frac{1}{2}(|0\rangle + |1\rangle)_A \otimes (|0\rangle + |1\rangle)_B = |+\rangle_A \otimes |+\rangle_B$$

Separable - this is a product state.

Conclusion: None of these states are entangled. All three can be factored into tensor products of single-qubit states.

[← Back to Exercise 2.1](#)

C.2.2. Solution 2.2: Schmidt Decomposition

Given: $|\psi\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$

From Exercise 2.1(c), we showed this factors as:

$$|\psi\rangle = |+\rangle_A \otimes |+\rangle_B$$

Schmidt decomposition:

$$|\psi\rangle = 1 \cdot |+\rangle_A \otimes |+\rangle_B$$

Schmidt rank: $r = 1$

Entanglement entropy:

C. Exercise Solutions

Since there's only one Schmidt coefficient ($\lambda_1 = 1$):

$$S = -1^2 \log_2(1^2) = -1 \cdot 0 = 0$$

Result: $S = 0$ bits, confirming the state is separable (not entangled).

[← Back to Exercise 2.2](#)

C.2.3. Solution 2.3: Entanglement Entropy

Given: $|\psi\rangle = \frac{2}{\sqrt{5}}|00\rangle + \frac{1}{\sqrt{5}}|11\rangle$

This is already in Schmidt form with:

$$\lambda_1 = \frac{2}{\sqrt{5}}, \quad \lambda_2 = \frac{1}{\sqrt{5}}$$

Verification of normalization:

$$\lambda_1^2 + \lambda_2^2 = \frac{4}{5} + \frac{1}{5} = 1 \quad \checkmark$$

Entanglement entropy:

$$S = -\lambda_1^2 \log_2(\lambda_1^2) - \lambda_2^2 \log_2(\lambda_2^2)$$

$$S = -\frac{4}{5} \log_2\left(\frac{4}{5}\right) - \frac{1}{5} \log_2\left(\frac{1}{5}\right)$$

$$S = -0.8 \log_2(0.8) - 0.2 \log_2(0.2)$$

$$S = -0.8 \times (-0.322) - 0.2 \times (-2.322)$$

$$S = 0.258 + 0.464 = 0.722 \text{ bits}$$

Is it maximally entangled?

No. For 2 qubits, maximal entanglement gives $S = 1$ bit (achieved by Bell states with equal Schmidt coefficients $\lambda_1 = \lambda_2 = \frac{1}{\sqrt{2}}$).

This state has $S \approx 0.722$ bits, so it is **partially entangled**.

[← Back to Exercise 2.3](#)

C.2.4. Solution 2.4: Bell State Transformations

Starting with $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$:

To get $|\Phi^-\rangle$: Apply Z to qubit B

$$\begin{aligned} (I \otimes Z)|\Phi^+\rangle &= \frac{1}{\sqrt{2}}(|0\rangle Z|0\rangle + |1\rangle Z|1\rangle) \\ &= \frac{1}{\sqrt{2}}(|0\rangle|0\rangle + |1\rangle(-|1\rangle)) = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) = |\Phi^-\rangle \quad \checkmark \end{aligned}$$

To get $|\Psi^+\rangle$: Apply X to qubit B

$$(I \otimes X)|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|0\rangle X|0\rangle + |1\rangle X|1\rangle)$$

C. Exercise Solutions

$$= \frac{1}{\sqrt{2}}(|0\rangle|1\rangle + |1\rangle|0\rangle) = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) = |\Psi^+\rangle \quad \checkmark$$

To get $|\Psi^-\rangle$: Apply Z then X to qubit B (or X then Z)

$$(I \otimes Z)|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|0\rangle Z|1\rangle + |1\rangle Z|0\rangle)$$

$$= \frac{1}{\sqrt{2}}(|0\rangle(-|1\rangle) + |1\rangle|0\rangle) = \frac{1}{\sqrt{2}}(-|01\rangle + |10\rangle)$$

$$= -\frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) = -|\Psi^-\rangle$$

The global phase -1 is physically irrelevant, so this equals $|\Psi^-\rangle$. $\checkmark$

Summary:

Target State	Gate on B
$ \Phi^+\rangle$	I (identity)
$ \Phi^-\rangle$	Z
$ \Psi^+\rangle$	X
$ \Psi^-\rangle$	XZ (or ZX)

[← Back to Exercise 2.4](#)

C.2.5. Solution 2.5: Partial Measurement

Given: $|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$

(a) Measure qubit A in Z-basis, outcome 0:

The state has two terms:

- $|01\rangle$: qubit A is in $|0\rangle$ ✓ (matches outcome)
- $|10\rangle$: qubit A is in $|1\rangle$ ✗ (doesn't match)

Only the $|01\rangle$ term survives. The post-measurement state is:

$$|\psi_{\text{post}}\rangle = |01\rangle = |0\rangle_A \otimes |1\rangle_B$$

(b) Probability of measuring qubit B as 1:

After the measurement in part (a), qubit B is in state $|1\rangle$ with certainty.

$$P(B = 1) = 1$$

This demonstrates the perfect anti-correlation in $|\Psi^+\rangle$: if A is measured as 0, B must be 1, and vice versa.

[← Back to Exercise 2.5](#)

C.2.6. Solution 2.6: Creating Bell States

Initial state: $|00\rangle$

Step 1: Apply Hadamard gate H to qubit A

$$\begin{aligned} H|0\rangle_A \otimes |0\rangle_B &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)_A \otimes |0\rangle_B \\ &= \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) \end{aligned}$$

Step 2: Apply CNOT gate (A = control, B = target)

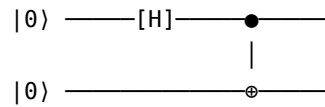
The CNOT gate flips qubit B if qubit A is $|1\rangle$:

- $\text{CNOT}|00\rangle = |00\rangle$ (A=0, B unchanged)
- $\text{CNOT}|10\rangle = |11\rangle$ (A=1, B flipped)

Applying to our state:

$$\text{CNOT} \left[\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) \right] = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = |\Phi^+\rangle \quad \checkmark$$

Circuit diagram:



This is the standard Bell state preparation circuit, producing $|\Phi^+\rangle$.

[← Back to Exercise 2.6](#)

C.3. Chapter 3: n-Qubit Systems

C.3.1. Solution 3.1: GHZ State Verification

For $|\text{GHZ}_4\rangle = \frac{1}{\sqrt{2}}(|0000\rangle + |1111\rangle)$:

(a) Probability of measuring outcome $|0101\rangle$:

The GHZ state only contains the basis states $|0000\rangle$ and $|1111\rangle$. The state $|0101\rangle$ has zero amplitude:

$$P(0101) = |\langle 0101 | \text{GHZ}_4 \rangle|^2 = 0$$

This is a key feature of GHZ states: only “all-zeros” or “all-ones” outcomes are possible.

(b) Measuring qubits 1 and 2 with outcomes 0 and 1:

Check which terms in the GHZ state match this outcome:

- $|0000\rangle$: qubit 1 = 0 ✓, qubit 2 = 0 ✗
- $|1111\rangle$: qubit 1 = 1 ✗, qubit 2 = 1 ✓

Neither term matches! This measurement outcome is impossible for a GHZ state.

$$P(\text{qubit 1} = 0, \text{qubit 2} = 1) = 0$$

This demonstrates the perfect correlation in GHZ states: all qubits must have the same value.

[← Back to Exercise 3.1](#)

C. Exercise Solutions

C.3.2. Solution 3.2: W State Properties

For $|W_4\rangle = \frac{1}{2}(|1000\rangle + |0100\rangle + |0010\rangle + |0001\rangle)$:

(a) Probability of measuring $|1000\rangle$:

$$P(1000) = \left| \frac{1}{2} \right|^2 = \frac{1}{4}$$

By symmetry, each of the four basis states has probability $\frac{1}{4}$.

(b) After measuring qubit 1 and getting 0:

Terms with qubit 1 = 0: $|0100\rangle, |0010\rangle, |0001\rangle$

Probability of outcome 0:

$$P(\text{qubit 1} = 0) = 3 \times \left| \frac{1}{2} \right|^2 = \frac{3}{4}$$

Unnormalized post-measurement state:

$$\frac{1}{2}(|0100\rangle + |0010\rangle + |0001\rangle)$$

Normalizing (divide by $\sqrt{P(0)} = \sqrt{3/4} = \frac{\sqrt{3}}{2}$):

$$|\psi_{\text{post}}\rangle = \frac{1}{\sqrt{3}}(|0100\rangle + |0010\rangle + |0001\rangle)$$

$$= |0\rangle_1 \otimes \frac{1}{\sqrt{3}}(|100\rangle + |010\rangle + |001\rangle)_{234}$$

$$= |0\rangle_1 \otimes |W_3\rangle_{234} \quad \checkmark$$

This demonstrates the robustness of W states: losing one qubit (with outcome 0) preserves W-type entanglement in the remaining qubits.

← Back to Exercise 3.2

C.3.3. Solution 3.3: Entanglement Entropy Calculation

For $|W_3\rangle = \frac{1}{\sqrt{3}}(|100\rangle + |010\rangle + |001\rangle)$ with partition 12|3:

Step 1: Rewrite in terms of the bipartition

Group by qubit 3's value:

$$\begin{aligned} |W_3\rangle &= \frac{1}{\sqrt{3}}|10\rangle_{12}|0\rangle_3 + \frac{1}{\sqrt{3}}|01\rangle_{12}|0\rangle_3 + \frac{1}{\sqrt{3}}|00\rangle_{12}|1\rangle_3 \\ &= \frac{1}{\sqrt{3}}(|10\rangle + |01\rangle)_{12} \otimes |0\rangle_3 + \frac{1}{\sqrt{3}}|00\rangle_{12} \otimes |1\rangle_3 \end{aligned}$$

Step 2: Find Schmidt decomposition

Normalize the first term's coefficient on system 12:

$$|W_3\rangle = \sqrt{\frac{2}{3}} \cdot \frac{1}{\sqrt{2}}(|10\rangle + |01\rangle)_{12} \otimes |0\rangle_3 + \frac{1}{\sqrt{3}}|00\rangle_{12} \otimes |1\rangle_3$$

Schmidt coefficients:

$$\lambda_1 = \sqrt{\frac{2}{3}}, \quad \lambda_2 = \frac{1}{\sqrt{3}} = \sqrt{\frac{1}{3}}$$

Step 3: Calculate entropy

C. Exercise Solutions

$$\lambda_1^2 = \frac{2}{3}, \quad \lambda_2^2 = \frac{1}{3}$$

$$S = -\frac{2}{3} \log_2 \left(\frac{2}{3} \right) - \frac{1}{3} \log_2 \left(\frac{1}{3} \right)$$

$$S = -\frac{2}{3}(\log_2 2 - \log_2 3) - \frac{1}{3}(-\log_2 3)$$

$$S = -\frac{2}{3}(1 - 1.585) + \frac{1}{3}(1.585)$$

$$S = -\frac{2}{3}(-0.585) + 0.528 = 0.390 + 0.528 \approx 0.918 \text{ bits}$$

Result: $S \approx 0.918$ bits, which is less than 1 bit (not maximally entangled).

[← Back to Exercise 3.3](#)

C.3.4. Solution 3.4: LOCC Impossibility

Why can't we convert $|\text{GHZ}_3\rangle$ to $|W_3\rangle$ using LOCC?

Key insight: LOCC operations can only *decrease* (or preserve) entanglement measures, never increase them.

Argument 1: Different bipartite entanglement structure

- GHZ has $S = 1$ bit for *all* bipartitions (1|23, 2|13, 3|12)
- W has $S \approx 0.918$ bits for all bipartitions

If we could convert $\text{GHZ} \rightarrow \text{W}$ via LOCC, we would be *decreasing* entropy, which is allowed. But we also cannot convert $\text{W} \rightarrow \text{GHZ}$, because that would require *increasing* entropy.

Argument 2: Robustness under qubit loss

- GHZ: Losing any qubit leaves the other two in a *product state* (zero entanglement)
- W: Losing any qubit (with outcome 0) leaves the other two in a *Bell state* (maximal 2-qubit entanglement)

LOCC cannot change this fundamental behavior. If GHZ could become W, then losing a qubit from the “converted” state should behave like W, but it doesn’t.

Argument 3: Genuine vs distributed entanglement

GHZ has “all-or-nothing” correlations: measuring any qubit determines all others. W has “distributed” correlations: information is spread across all qubits. These are fundamentally incompatible structures that LOCC cannot interconvert.

Conclusion: GHZ and W represent different *entanglement classes* that are LOCC-inequivalent.

[← Back to Exercise 3.4](#)

C.3.5. Solution 3.5: Multipartite Correlation

For $|\psi\rangle = \frac{1}{2}(|000\rangle + |011\rangle + |101\rangle + |110\rangle)$:

(a) Check all bipartitions for entanglement:

C. Exercise Solutions

Partition 1|23:

$$\begin{aligned}
 |\psi\rangle &= \frac{1}{2}|0\rangle(|00\rangle + |11\rangle) + \frac{1}{2}|1\rangle(|01\rangle + |10\rangle) \\
 &= \frac{1}{\sqrt{2}}|0\rangle \cdot \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) + \frac{1}{\sqrt{2}}|1\rangle \cdot \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \\
 &= \frac{1}{\sqrt{2}}|0\rangle|\Phi^+\rangle + \frac{1}{\sqrt{2}}|1\rangle|\Psi^+\rangle
 \end{aligned}$$

Schmidt rank = 2 $\rightarrow$ **Entangled** ✓

Partition 2|13:

Grouping by qubit 2:

- Qubit 2 = 0: $|000\rangle, |101\rangle \rightarrow$ qubits 1,3 in $(|00\rangle + |11\rangle)$
- Qubit 2 = 1: $|011\rangle, |110\rangle \rightarrow$ qubits 1,3 in $(|01\rangle + |10\rangle)$

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle_2|\Phi^+\rangle_{13} + \frac{1}{\sqrt{2}}|1\rangle_2|\Psi^+\rangle_{13}$$

Schmidt rank = 2 $\rightarrow$ **Entangled** ✓

Partition 3|12:

Grouping by qubit 3:

- Qubit 3 = 0: $|000\rangle, |110\rangle \rightarrow$ qubits 1,2 in $(|00\rangle + |11\rangle)$
- Qubit 3 = 1: $|011\rangle, |101\rangle \rightarrow$ qubits 1,2 in $(|01\rangle + |10\rangle)$

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle_3|\Phi^+\rangle_{12} + \frac{1}{\sqrt{2}}|1\rangle_3|\Psi^+\rangle_{12}$$

Schmidt rank = 2 $\rightarrow$ **Entangled** ✓

Conclusion: Yes, $|\psi\rangle$ is **genuinely tripartite entangled** (entangled across all bipartitions).

(b) Entanglement entropy for partition 1|23:

From part (a), the Schmidt coefficients are $\lambda_1 = \lambda_2 = \frac{1}{\sqrt{2}}$.

$$S = -\frac{1}{2}\log_2 \frac{1}{2} - \frac{1}{2}\log_2 \frac{1}{2} = 1 \text{ bit}$$

This state has **maximal bipartite entanglement** for partition 1|23.

[← Back to Exercise 3.5](#)

C.3.6. Solution 3.6: Fragility vs Robustness

Experimental Design: Comparing GHZ and W State Robustness

Setup:

1. Prepare many copies of $|\text{GHZ}_3\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$
2. Prepare many copies of $|\text{W}_3\rangle = \frac{1}{\sqrt{3}}(|100\rangle + |010\rangle + |001\rangle)$
3. Distribute qubits to three parties: Alice (qubit 1), Bob (qubit 2), Charlie (qubit 3)

C. Exercise Solutions

Experiment: Simulate qubit loss by measuring qubit 1

Alice measures her qubit in the Z-basis, then Bob and Charlie test whether their qubits are still entangled.

For GHZ states:

- Alice measures 0 $\rightarrow$ Bob and Charlie have $|00\rangle$ (product state)
- Alice measures 1 $\rightarrow$ Bob and Charlie have $|11\rangle$ (product state)
- **Result:** Bob and Charlie share NO entanglement, regardless of Alice's outcome

For W states:

- Alice measures 1 (prob. $1/3$) $\rightarrow$ Bob and Charlie have $|00\rangle$ (product state)
- Alice measures 0 (prob. $2/3$) $\rightarrow$ Bob and Charlie have $\frac{1}{\sqrt{2}}(|10\rangle + |01\rangle) = |\Psi^+\rangle$ (Bell state!)
- **Result:** With probability $2/3$, Bob and Charlie still share maximal entanglement

Verification:

Bob and Charlie perform Bell inequality tests (CHSH) on their remaining qubits:

- GHZ remnants: Always satisfy classical bounds (no entanglement)
- W remnants: Violate Bell inequality $2/3$ of the time (quantum entanglement preserved)

Conclusion: This experiment directly demonstrates that W states are robust to qubit loss while GHZ states are fragile.

[← Back to Exercise 3.6](#)

C.3.7. Solution 3.7: n -Qubit Generalization

General W state formula:

$$|W_n\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^n |0 \cdots 0 \underset{i\text{-th position}}{1} 0 \cdots 0\rangle$$

This is an equal superposition of all n basis states with exactly one qubit in $|1\rangle$.

Claim: Measuring k qubits and getting all outcomes 0 leaves the remaining $n - k$ qubits in state $|W_{n-k}\rangle$.

Proof:

Let qubits $1, 2, \dots, k$ be measured, and qubits $k + 1, \dots, n$ remain.

The $|W_n\rangle$ state has n terms. Each term has exactly one qubit in state $|1\rangle$:

- Terms where the $|1\rangle$ is in positions $1, 2, \dots, k$: These have at least one measured qubit = 1, so they don't survive the "all zeros" measurement.
- Terms where the $|1\rangle$ is in positions $k + 1, \dots, n$: These have all measured qubits = 0, so they survive.

Number of surviving terms: $n - k$

Unnormalized post-measurement state:

$$\frac{1}{\sqrt{n}} \sum_{i=k+1}^n |0\rangle^{\otimes k} \otimes |0 \cdots 1_i \cdots 0\rangle$$

Probability of all-zeros outcome:

$$P(0^k) = \frac{n - k}{n}$$

C. Exercise Solutions

Normalized post-measurement state:

$$|\psi_{\text{post}}\rangle = \frac{1}{\sqrt{n-k}} \sum_{i=k+1}^n |0 \dots 1_i \dots 0\rangle = |W_{n-k}\rangle \quad \square$$

Physical interpretation: The W state “stores” one unit of excitation distributed across n qubits. Measuring k qubits and finding them all in $|0\rangle$ confirms the excitation is among the remaining $n - k$ qubits, which are now in $|W_{n-k}\rangle$.

Corollary: This process can continue until only 2 qubits remain, yielding:

$$|W_2\rangle = \frac{1}{\sqrt{2}}(|10\rangle + |01\rangle) = |\Psi^+\rangle$$

A Bell state! This is why W states are robust: entanglement persists down to the last pair.

[← Back to Exercise 3.7](#)

C.4. Chapter 4: MBQC Foundations

C.4.1. Solution 4.1: Cluster State Preparation

For a 2-qubit cluster state on graph 1—2:

(a) Initial state $|+\rangle^{\otimes 2}$:

$$|+\rangle^{\otimes 2} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

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

(b) Apply CZ_{12} :

The controlled-Z gate acts as: $CZ|ab\rangle = (-1)^{a \cdot b}|ab\rangle$

- $CZ|00\rangle = (-1)^0|00\rangle = |00\rangle$
- $CZ|01\rangle = (-1)^0|01\rangle = |01\rangle$
- $CZ|10\rangle = (-1)^0|10\rangle = |10\rangle$
- $CZ|11\rangle = (-1)^1|11\rangle = -|11\rangle$

Resulting cluster state:

$$|\text{Cluster}\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$$

(c) Verify entanglement by attempting to factor:

Assume $|\text{Cluster}\rangle = (\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle)$

Expanding: $\alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle$

Comparing coefficients with $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$:

$$\alpha\gamma = \frac{1}{2}, \quad \alpha\delta = \frac{1}{2}, \quad \beta\gamma = \frac{1}{2}, \quad \beta\delta = -\frac{1}{2}$$

From the first three equations: $\alpha\gamma = \alpha\delta$ implies $\gamma = \delta$, and $\alpha\gamma = \beta\gamma$ implies $\alpha = \beta$.

But if $\alpha = \beta$ and $\gamma = \delta$, then:

$$\beta\delta = \alpha\gamma = \frac{1}{2} \neq -\frac{1}{2} \quad \not\Leftarrow \text{Contradiction!}$$

Therefore, the state **cannot be factored** and is **entangled**. $\square$

[← Back to Exercise 4.1](#)

C.4.2. Solution 4.2: Measurement Outcomes

For the 3-qubit chain cluster state, we first derive its explicit form.

Cluster state preparation:

Starting with $|+\rangle^{\otimes 3}$ and applying CZ_{12} then CZ_{23} :

The coefficient of $|abc\rangle$ becomes $\frac{1}{2\sqrt{2}}(-1)^{ab+bc}$

Computing each term:

$ abc\rangle$	$ab + bc$	$(-1)^{ab+bc}$
$ 000\rangle$	0	+1
$ 001\rangle$	0	+1
$ 010\rangle$	0	+1
$ 011\rangle$	1	-1
$ 100\rangle$	0	+1
$ 101\rangle$	0	+1
$ 110\rangle$	1	-1
$ 111\rangle$	2	+1

$$|\text{Cluster}\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle - |011\rangle + |100\rangle + |101\rangle - |110\rangle + |111\rangle)$$

Grouping by qubit 1:

$$|\text{Cluster}\rangle = \frac{1}{2\sqrt{2}} [|0\rangle(|00\rangle + |01\rangle + |10\rangle - |11\rangle) + |1\rangle(|00\rangle + |01\rangle - |10\rangle + |11\rangle)]$$

Measuring qubit 1 in X-basis, outcome 0 (i.e., $|+\rangle$):

Project onto $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$:

$$\begin{aligned}\langle +|_1|\text{Cluster}\rangle &\propto (|00\rangle + |01\rangle + |10\rangle - |11\rangle) + (|00\rangle + |01\rangle - |10\rangle + |11\rangle) \\ &= 2|00\rangle + 2|01\rangle = 2(|00\rangle + |01\rangle)\end{aligned}$$

Normalizing:

$$|\psi\rangle_{23} = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle) = |0\rangle_2 \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)_3 = |0\rangle_2|+\rangle_3$$

Answer: After measuring qubit 1 in X-basis with outcome 0, qubits 2-3 are in state $|0\rangle|+\rangle$.

[← Back to Exercise 4.2](#)

C.4.3. Solution 4.3: Gate Teleportation

Standard teleportation:

1. Alice has state $|\psi\rangle$ on qubit 1
2. Bell pair $|\Phi^+\rangle$ shared on qubits 2 (Alice) and 3 (Bob)
3. Alice performs Bell measurement on qubits 1-2, getting outcome (a, b)
4. Bob applies $X^a Z^b$ to qubit 3
5. Result: $|\psi\rangle$ appears on qubit 3

C. Exercise Solutions

How Hadamard-basis measurement applies H:

The key insight is that a Bell measurement can be decomposed as:

- Apply $CNOT_{12}$ (qubit 1 controls qubit 2)
- Apply H to qubit 1
- Measure both qubits in computational (Z) basis

If Alice instead measures qubit 1 in the Hadamard basis (while still doing the CNOT), the sequence becomes:

1. $CNOT_{12}$
2. Measure qubit 1 in X-basis (equivalent to: H , then measure in Z-basis)
3. Measure qubit 2 in Z-basis

The change in measurement basis propagates through the teleportation protocol. Since $H^2 = I$, measuring in the H -rotated basis effectively “inserts” an H gate into the teleported state.

Mathematical explanation:

In standard teleportation, the combined state before measurement is:

$$|\psi\rangle_1 |\Phi^+\rangle_{23} = |\psi\rangle_1 \cdot \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)_{23}$$

After CNOT and measuring qubit 1 in X-basis instead of Z-basis, the effective operation on Bob’s qubit includes an additional Hadamard:

$$|\psi\rangle \xrightarrow{\text{teleport with H-basis}} H|\psi\rangle \text{ (up to Pauli corrections)}$$

Physical intuition: The measurement basis choice determines which “frame” we observe the quantum information in. Rotating the measurement basis by H rotates the output by H .

← [Back to Exercise 4.3](#)

C.4.4. Solution 4.4: Pattern Design

Goal: Implement Hadamard gate H on a 3-qubit chain cluster state.

Pattern specification:

Graph: Linear chain 1—2—3

Protocol:

1. **Prepare** 3-qubit cluster state with input encoded on qubit 1
2. **Measure qubit 1** in X-basis ($\theta_1 = 0$):
 - Outcome: $s_1 \in \{0, 1\}$
3. **Measure qubit 2** in X-basis ($\theta_2 = 0$):
 - No adaptation needed for Hadamard
 - Outcome: $s_2 \in \{0, 1\}$
4. **Apply corrections to qubit 3:**

$$X^{s_2} Z^{s_1}$$

Result: Qubit 3 contains $H|\psi_{\text{in}}\rangle$

Explanation:

For a 3-qubit chain with both measurements in X-basis:

- The first measurement “teleports” the input through the cluster
- The second measurement applies the Hadamard transformation
- The byproduct operators X and Z depend on measurement outcomes

C. Exercise Solutions

Alternative (2-qubit pattern):

Hadamard can also be implemented on just 2 qubits:

1. Prepare 2-qubit cluster $|+\rangle|+\rangle$ with CZ
2. Encode input on qubit 1
3. Measure qubit 1 in Y-basis ($\theta = \pi/2$), outcome s
4. Apply Z^s to qubit 2

Result: $H|\psi\rangle$ on qubit 2

[← Back to Exercise 4.4](#)

C.4.5. Solution 4.5: Resource Scaling

Given:

- Circuit with depth d and width n requires approximately $d \cdot n$ qubits in MBQC
- Grover's algorithm on n qubits with $O(\sqrt{2^n})$ gates

Analysis:

Grover's algorithm parameters:

- **Width:** n qubits (the search space has 2^n elements)
- **Total gates:** $O(\sqrt{2^n}) = O(2^{n/2})$

To estimate depth, we need to consider parallelization:

- Each Grover iteration applies an oracle and a diffusion operator
- Number of iterations: $O(\sqrt{2^n})$
- Gates per iteration: $O(n)$ (oracle and diffusion each use $O(n)$ gates)
- But wait, the problem states total gates is $O(\sqrt{2^n})$, which suggests:

- Either very few iterations, or
- The oracle is very simple (constant-depth)

Case 1: If depth $\approx$ total gates (sequential)

Cluster size $\approx n \cdot O(2^{n/2}) = O(n \cdot 2^{n/2})$

Case 2: If maximally parallelized

With n qubits and $O(2^{n/2})$ gates, minimum depth is:

$$d \geq \frac{O(2^{n/2})}{n}$$

Cluster size $\approx n \cdot \frac{2^{n/2}}{n} = O(2^{n/2}) = O(\sqrt{2^n})$

Practical estimate:

For typical Grover implementations:

- $O(\sqrt{2^n})$ iterations
- Each iteration has $O(\log n)$ to $O(n)$ depth

Answer: The cluster state size is approximately:

$$\boxed{O(n \cdot 2^{n/2})} \text{ to } \boxed{O(n^2 \cdot 2^{n/2})}$$

depending on the oracle complexity. For a simple oracle, the lower bound $O(2^{n/2})$ qubits may suffice.

[← Back to Exercise 4.5](#)

C.4.6. Solution 4.6: Advantage Analysis

Three scenarios where MBQC has advantages:

1. Photonic quantum computing

- Photons are easy to entangle (beam splitters, parametric down-conversion)
- Direct photon-photon gates are extremely difficult
- MBQC only requires single-photon measurements, which are routine
- Companies like PsiQuantum and Xanadu use MBQC for this reason

2. Fault-tolerant quantum computing with surface codes

- Surface codes are naturally implemented as MBQC on a 2D cluster
- Syndrome extraction maps to measurements on the cluster
- Topological protection emerges from the cluster state structure
- This is the leading approach for large-scale fault-tolerant QC

3. Blind quantum computing

- Client can delegate computation to a server without revealing the algorithm
- Client only needs to prepare single qubits and specify measurement angles
- Server prepares cluster state and measures, but learns nothing about the computation
- MBQC naturally separates “resource preparation” from “computation specification”

Three scenarios where the circuit model is preferable:

1. Superconducting and trapped-ion hardware

- These platforms have excellent two-qubit gates (99%+ fidelity)
- Preparing large cluster states requires many gates anyway
- Circuit model avoids the qubit overhead of MBQC (~5x more qubits)
- Most current quantum computers (IBM, Google, IonQ) use circuit model

2. Near-term (NISQ) algorithms

- Variational algorithms (VQE, QAOA) need parameter updates
- Circuit model allows direct gate parameter optimization
- MBQC would require recomputing measurement patterns each iteration
- Classical-quantum hybrid loops are simpler in circuit model

3. Algorithm development and simulation

- Circuit model has mature software ecosystem (Qiskit, Cirq, etc.)
- Quantum algorithms are typically designed in gate language
- Classical simulation of circuits is well-understood
- Educational resources predominantly use circuit model

← Back to Exercise 4.6

C.5. Chapter 5: Cluster States

C.5.1. Solution 5.1: Stabilizer Verification

Verify that $K_2 = Z_1 X_2$ stabilizes $|G\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$.

Step 1: Apply X_2 first

X_2 flips the second qubit:

C. Exercise Solutions

- $X_2|00\rangle = |01\rangle$
- $X_2|01\rangle = |00\rangle$
- $X_2|10\rangle = |11\rangle$
- $X_2|11\rangle = |10\rangle$

$$X_2|G\rangle = \frac{1}{2}(|01\rangle + |00\rangle + |11\rangle - |10\rangle)$$

Step 2: Apply Z_1

Z_1 applies phase -1 when qubit 1 is $|1\rangle$:

- $Z_1|00\rangle = |00\rangle$
- $Z_1|01\rangle = |01\rangle$
- $Z_1|10\rangle = -|10\rangle$
- $Z_1|11\rangle = -|11\rangle$

$$Z_1X_2|G\rangle = \frac{1}{2}(|01\rangle + |00\rangle - |11\rangle + |10\rangle)$$

Step 3: Reorder terms

$$K_2|G\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle) = |G\rangle \quad \checkmark$$

[← Back to Exercise 5.1](#)

C.5.2. Solution 5.2: 4-Qubit Chain Stabilizers

For the 4-qubit linear cluster state 1—2—3—4:

Neighbors:

- $N(1) = \{2\}$
- $N(2) = \{1, 3\}$
- $N(3) = \{2, 4\}$
- $N(4) = \{3\}$

Stabilizer generators using the formula $K_i = X_i \prod_{j \in N(i)} Z_j$:

$$\begin{aligned} K_1 &= X_1 Z_2 \\ K_2 &= Z_1 X_2 Z_3 \\ K_3 &= Z_2 X_3 Z_4 \\ K_4 &= Z_3 X_4 \end{aligned}$$

Verification: These four operators:

- Are all Hermitian (equal to their conjugate transpose)
- Square to identity: $K_i^2 = I$
- Mutually commute: $K_i K_j = K_j K_i$ for all i, j
- Uniquely determine the 4-qubit cluster state

[← Back to Exercise 5.2](#)

C.5.3. Solution 5.3: Stabilizer Commutation

Prove that $K_1 = X_1Z_2$ and $K_2 = Z_1X_2Z_3$ commute.

Method: Count anticommuting pairs

Two Pauli operators commute if they share an even number of positions where they anticommute.

Writing out the operators on each qubit:

Qubit	K_1	K_2	Commute?
1	X	Z	No (anticommute)
2	Z	X	No (anticommute)
3	I	Z	Yes (commute)

Number of anticommuting positions: 2 (even)

Therefore: K_1 and K_2 commute: $K_1K_2 = K_2K_1$ ✓

Alternative verification via direct calculation:

$$K_1K_2 = (X_1Z_2)(Z_1X_2Z_3) = (X_1Z_1)(Z_2X_2)(Z_3)$$

Using $XZ = -ZX$ (anticommutation):

- $X_1Z_1 = iY_1$
- $Z_2X_2 = -iY_2$

$$K_1K_2 = (iY_1)(-iY_2)(Z_3) = Y_1Y_2Z_3$$

$$K_2K_1 = (Z_1X_2Z_3)(X_1Z_2) = (Z_1X_1)(X_2Z_2)(Z_3)$$

- $Z_1 X_1 = -iY_1$
- $X_2 Z_2 = iY_2$

$$K_2 K_1 = (-iY_1)(iY_2)(Z_3) = Y_1 Y_2 Z_3$$

Result: $K_1 K_2 = K_2 K_1 = Y_1 Y_2 Z_3$ ✓

← Back to Exercise 5.3

C.5.4. Solution 5.4: Local Complementation

Starting with path graph 1—2—3—4, apply Hadamard to qubit 3.

Step 1: Identify neighbors of qubit 3

$$N(3) = \{2, 4\}$$

Step 2: Find induced subgraph on $N(3)$

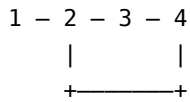
The induced subgraph on vertices $\{2, 4\}$ contains only those two vertices. In the original graph, there is **no edge** between 2 and 4.

Step 3: Complement the induced subgraph

Complement rule: toggle each edge (add if absent, remove if present).

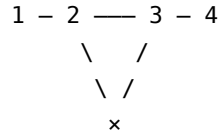
Since $(2, 4)$ is absent → **add edge** $(2, 4)$

Step 4: Resulting graph



C. Exercise Solutions

Or equivalently:



The new graph has edges: $(1, 2), (2, 3), (3, 4), (2, 4)$

State transformation: $H_3|G\rangle = |G'\rangle$ where G' is the new graph.

[← Back to Exercise 5.4](#)

C.5.5. Solution 5.5: Measurement Update

For 3-qubit chain 1—2—3 with stabilizers:

$$K_1 = X_1 Z_2, \quad K_2 = Z_1 X_2 Z_3, \quad K_3 = Z_2 X_3$$

Measure qubit 1 in X-basis with outcome 0.

Step 1: Understand the measurement

X-basis measurement with outcome 0 means qubit 1 is projected to $|+\rangle$.

This implies $X_1 = +1$ (eigenvalue of $|+\rangle$).

Step 2: Update stabilizers

From $K_1 = X_1 Z_2$ with $X_1 \rightarrow +1$:

- New stabilizer: Z_2

From $K_3 = Z_2 X_3$ (doesn't involve qubit 1):

- Remains: Z_2X_3

Step 3: Handle $K_2 = Z_1X_2Z_3$

Since qubit 1 is measured in X-basis, Z_1 becomes indeterminate. We multiply K_2 by K_1 to eliminate qubit 1:

$$K_2 \cdot K_1 = (Z_1X_2Z_3)(X_1Z_2) = Z_1X_1 \cdot X_2Z_2 \cdot Z_3$$

After measuring with $X_1 = +1$, the qubit-1 part contributes only a phase. The remaining stabilizer on qubits 2-3 is:

$$X_2Z_3 \quad (\text{up to sign depending on outcome})$$

Final stabilizers for qubits 2-3:

$$K'_2 = X_2Z_3, \quad K'_3 = Z_2X_3$$

These are exactly the stabilizers of a 2-qubit cluster state 2—3, which makes sense: measuring out qubit 1 in X-basis leaves the remaining qubits in a 2-qubit cluster state.

[← Back to Exercise 5.5](#)

C.5.6. Solution 5.6: Graph State Entanglement

For a star graph with central qubit 0 connected to n outer qubits $\{1, 2, \dots, n\}$.

Step 1: Derive the star graph state

$$|\text{star}\rangle = CZ_{01}CZ_{02} \dots CZ_{0n}|+\rangle^{\otimes(n+1)}$$

The central qubit starts in $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

When the center is $|0\rangle$: all CZ gates act as identity on outer qubits $\rightarrow$ outer qubits stay $|+\rangle^{\otimes n}$

When the center is $|1\rangle$: each CZ applies Z to the corresponding outer qubit $\rightarrow |+\rangle \xrightarrow{Z} |-\rangle$

Result:

$$|\text{star}\rangle = \frac{1}{\sqrt{2}} (|0\rangle|+\rangle^{\otimes n} + |1\rangle|-\rangle^{\otimes n})$$

Step 2: Find Schmidt decomposition

This is already in Schmidt form with partition (central qubit) | (outer qubits):

$$|\text{star}\rangle = \frac{1}{\sqrt{2}}|0\rangle|\phi_0\rangle + \frac{1}{\sqrt{2}}|1\rangle|\phi_1\rangle$$

where $|\phi_0\rangle = |+\rangle^{\otimes n}$ and $|\phi_1\rangle = |-\rangle^{\otimes n}$ are orthogonal.

Schmidt coefficients: $\lambda_1 = \lambda_2 = \frac{1}{\sqrt{2}}$

Step 3: Calculate entanglement entropy

$$S = - \sum_i \lambda_i^2 \log_2 \lambda_i^2 = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1 \text{ bit}$$

Answer: The entanglement entropy between the central qubit and all outer qubits is **1 bit**, regardless of n .

Physical interpretation: The star graph state is essentially a GHZ-like state where the central qubit is maximally entangled with the collective state of all outer qubits.

← Back to Exercise 5.6

C.5.7. Solution 5.7: Resource Comparison

For a 2D cluster state simulating a circuit of depth d on n qubits, the cluster size is approximately $O(d \times n)$.

(a) Grover's algorithm on 10 qubits

Parameters:

- Search space: $2^{10} = 1024$ elements
- Number of Grover iterations: $O(\sqrt{2^{10}}) = O(32)$
- Gates per iteration: $O(n) = O(10)$ (oracle + diffusion)
- Total depth: $d \approx 32 \times 10 = 320$
- Width: $n = 10$ qubits

2D cluster size:

$$\text{Cluster qubits} \approx d \times n = 320 \times 10 = 3,200 \text{ qubits}$$

(b) Quantum Fourier Transform on 20 qubits

Parameters:

- Width: $n = 20$ qubits
- Sequential depth: $O(n^2) = O(400)$ (naive implementation)

C. Exercise Solutions

- Parallelized depth: $O(n) = O(20)$ (with sufficient ancillas)

2D cluster size (sequential):

$$\text{Cluster qubits} \approx 400 \times 20 = 8,000 \text{ qubits}$$

2D cluster size (parallelized):

$$\text{Cluster qubits} \approx 20 \times 20 = 400 \text{ qubits}$$

Summary:

Algorithm	Qubits	Depth	Cluster Size
Grover (10 qubits)	10	~320	~3,200
QFT (20 qubits, sequential)	20	~400	~8,000
QFT (20 qubits, parallel)	20	~20	~400

Note: These are rough estimates. Actual implementations may have constant-factor overhead for error correction, gate decomposition, and MBQC-specific requirements.

[← Back to Exercise 5.7](#)

C.6. Chapter 6: Measurement Patterns

C.6.1. Solution 6.1: Hadamard Compilation

Complete measurement pattern for the Hadamard gate:

Cluster preparation:

C.6. Chapter 6: Measurement Patterns

```
N_1      # Create qubit 1 in state |+⟩ (this will hold input)
N_2      # Create qubit 2 in state |+⟩ (this will be output)
E_12     # Entangle qubits 1 and 2 with CZ
```

Measurement:

```
M_1^0 → s # Measure qubit 1 in X-basis ( $\theta = 0$ ), outcome  $s \in \{0,1\}$ 
```

Correction:

```
Z_2^s     # Apply Z to qubit 2 if  $s = 1$ 
```

Summary:

Step	Command	Description
1	N_1, N_2	Create two qubits in $ +\rangle$
2	E_{12}	Apply CZ to create cluster
3	$M_1^0 \rightarrow s$	Measure qubit 1 in X-basis
4	Z_2^s	Apply Z correction if $s = 1$

Result: Qubit 2 contains $H|\psi\rangle$ where $|\psi\rangle$ was encoded in qubit 1.

Verification: This works because measuring in the X-basis ($|+\rangle, |-\rangle$) on a cluster state effectively applies a Hadamard transformation to the information as it “flows” through the cluster.

[← Back to Exercise 6.1](#)

C.6.2. Solution 6.2: Rotation Angle

Question: For rotation $R_Z(\theta)$ implemented by measuring in basis $|\pm_\alpha\rangle$, what is the relationship between θ and α ?

Answer: The measurement angle equals the rotation angle:

$$\boxed{\alpha = \theta}$$

Explanation:

The measurement basis states are:

$$|\pm_\alpha\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm e^{i\alpha}|1\rangle)$$

When measuring in this basis on a cluster state, the gate applied is:

$$R_Z(\alpha) = e^{-i\alpha Z/2} = \begin{pmatrix} e^{-i\alpha/2} & 0 \\ 0 & e^{i\alpha/2} \end{pmatrix}$$

Key insight: The phase factor $e^{i\alpha}$ in the measurement basis directly translates to the rotation angle in the implemented gate.

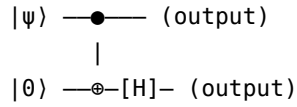
Special cases:

- $\alpha = 0$: X-basis measurement $\rightarrow R_Z(0) = I$ (identity, but with Hadamard effect from cluster)
- $\alpha = \pi/2$: Y-basis measurement $\rightarrow R_Z(\pi/2) = S$ (phase gate)
- $\alpha = \pi$: $-X$ -basis measurement $\rightarrow R_Z(\pi) = Z$

[← Back to Exercise 6.2](#)

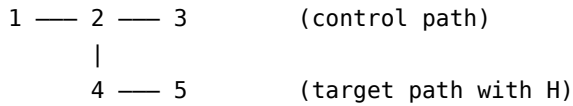
C.6.3. Solution 6.3: CNOT Pattern

Circuit:



This is CNOT followed by Hadamard on the target qubit.

Cluster graph:



Where:

- Qubit 1: Control input (encoded with $|\psi\rangle$)
- Qubit 4: Target input (initialized to $|+\rangle$, represents $|0\rangle$ in computational basis)
- Qubit 2: CNOT interaction ancilla
- Qubit 3: Control output
- Qubit 5: Target output (after CNOT and H)

Measurement pattern:

Qubit	Measurement Basis	Outcome
1	X-basis ($\theta = 0$)	s_1
2	X-basis ($\theta = 0$)	s_2
4	X-basis ($\theta = 0$)	s_4

Corrections:

C. Exercise Solutions

- On qubit 3 (control output): Z^{s_1}
- On qubit 5 (target output): $X^{s_2}Z^{s_4}$

Pattern commands:

```
N_1 N_2 N_3 N_4 N_5    # Create 5 qubits
E_12 E_23 E_24 E_45    # Create cluster structure
M_1^0 → s_1            # Measure control input
M_2^0 → s_2            # Measure CNOT ancilla
M_4^0 → s_4            # Measure target input (also implements H)
Z_3^{s_1}              # Correct control output
X_5^{s_2} Z_5^{s_4}    # Correct target output
```

Note: The H gate on the target is “absorbed” into the measurement of qubit 4 - measuring in X-basis naturally applies H to the information flow.

← [Back to Exercise 6.3](#)

C.6.4. Solution 6.4: Adaptive Measurement

Circuit: H followed by $R_Z(\theta)$

Question: If the first measurement (for H) gives outcome $s_1 = 1$, should the second measurement angle be θ or $\theta + \pi$?

Answer: The second measurement angle should be $\theta + \pi$.

$$\theta' = \theta + \pi \cdot s_1 = \theta + \pi$$

Justification:

When the first measurement gives outcome $s_1 = 1$, there is a **byproduct operator** that propagates through the computation. Specifically:

1. **After first measurement** ($s_1 = 1$): The state acquires an X byproduct operator.
2. **Commutation relation:** When X commutes past $R_Z(\theta)$:

$$X \cdot R_Z(\theta) = R_Z(-\theta) \cdot X$$

This is because $XZX = -Z$, so the rotation direction flips.

3. **Compensation:** To get the correct $R_Z(\theta)$ instead of $R_Z(-\theta)$, we need to change our measurement angle. Since:

$$R_Z(\theta + \pi) = -R_Z(\theta) \cdot Z$$

and the sign and Z can be absorbed into later corrections, measuring at $\theta + \pi$ effectively compensates for the sign flip.

4. **Adaptation rule:**

$$\theta_{\text{actual}} = \theta + \pi \cdot s_1$$

- If $s_1 = 0$: measure at θ (no adaptation needed)
- If $s_1 = 1$: measure at $\theta + \pi$ (flip the angle)

This is the essence of **feed-forward** in MBQC: measurement outcomes from earlier qubits determine the measurement bases for later qubits.

[← Back to Exercise 6.4](#)

C.6.5. Solution 6.5: Pattern Optimization

Problem: A depth-10 circuit on 5 qubits naively compiles to a $5 \times 11 = 55$ qubit cluster. Describe two techniques to reduce this.

Technique 1: Gate Cancellation

Simplify the circuit *before* compiling to MBQC:

- **Adjacent inverse gates:** $H \cdot H = I$, $CNOT \cdot CNOT = I$, $S \cdot S^\dagger = I$
- **Commutation:** Reorder gates that commute to enable more cancellations
- **Gate synthesis:** Replace sequences with equivalent shorter sequences

Example: If the circuit contains $H \cdot H$ on some qubit, removing these saves 2 layers $\rightarrow$ cluster reduced by $5 \times 2 = 10$ qubits.

Technique 2: Using Native CZ Gates

Replace $CNOT$ with CZ where possible:

- $CNOT = (I \otimes H) \cdot CZ \cdot (I \otimes H)$
- But if there are adjacent H gates, they may cancel
- CZ is **native** to cluster states (no ancilla needed for the entangling operation itself)

Savings: Each $CNOT$ replaced by CZ can save 1-2 ancilla qubits.

Additional techniques:

3. **Parallel measurement:** Gates on different qubits can share cluster layers
4. **Signal shifting:** Track Pauli corrections classically rather than implementing physically
5. **Lazy correction:** Defer corrections to the end, absorbing them into the final measurement interpretation

Estimated savings: With aggressive optimization, a 55-qubit naive cluster can often be reduced to **25-35 qubits** for typical circuits.

← Back to Exercise 6.5

C.6.6. Solution 6.6: Deterministic Gates

Claim: The Pauli X gate can be implemented deterministically in MBQC.

Proof:

In MBQC, Pauli gates (X , Y , Z) are special because they are **byproduct operators** - they arise naturally from measurement outcomes and are tracked classically rather than implemented physically.

Method 1: Classical tracking

The X gate doesn't require any measurement to implement. Instead:

1. When compiling, note that X should be applied
2. Track this as a “pending X correction” on that qubit
3. At the end of computation, either:
 - Apply the physical X gate, or
 - Flip the interpretation of the final measurement outcome

This is **deterministic** because no probabilistic measurement is involved.

Method 2: Measurement-based implementation

If we insist on using measurements:

1. **Prepare:** 2-qubit cluster with input $|\psi\rangle$ on qubit 1
2. **Measure:** Qubit 1 in the Z -basis

C. Exercise Solutions

3. Result:

- Outcome 0: Qubit 2 is in state $H|\psi\rangle \cdot |+\rangle$ projection
- Outcome 1: Qubit 2 is in state $HX|\psi\rangle \cdot |-\rangle$ projection

The key insight: although the outcome is random, the **correction needed is always the same up to the known measurement result**. We can predict exactly what correction is needed.

Why this is “deterministic”:

The term “deterministic” here means:

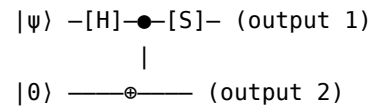
- The correction formula is **fixed** (not dependent on the outcome in a complicated way)
- The final state after correction is **always** $X|\psi\rangle$
- There’s no adaptation or feed-forward uncertainty

Final state = $X|\psi\rangle$ (always, regardless of measurement outcome)

[← Back to Exercise 6.6](#)

C.6.7. Solution 6.7: Circuit to Pattern

Circuit:

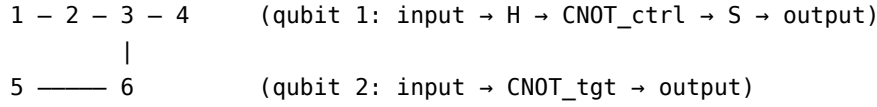


where $S = R_Z(\pi/2)$ is the phase gate.

Step 1: Identify gates

- H on qubit 1
- $CNOT$ with qubit 1 as control, qubit 2 as target
- $S = R_Z(\pi/2)$ on qubit 1

Step 2: Cluster graph



Vertex roles:

- 1: Input qubit 1 (holds $|\psi\rangle$)
- 2: Ancilla for H gate
- 3: CNOT interaction vertex (connects control and target paths)
- 4: Output qubit 1 (also implements S via measurement of vertex 3)
- 5: Input qubit 2 (initialized in cluster as $|+\rangle$)
- 6: Output qubit 2

Step 3: Measurement pattern

Qubit	Role	Measurement Basis	Outcome
1	H gate	X-basis ($\theta = 0$)	s_1
2	Flow	X-basis ($\theta = 0$), adapted	s_2
3	CNOT + S	$\theta = \pi/4$ (for S), adapted	s_3
5	CNOT target	X-basis ($\theta = 0$)	s_5

Adaptive angles:

- $\theta'_2 = 0 + \pi \cdot s_1$

C. Exercise Solutions

- $\theta'_3 = \pi/4 + \pi \cdot s_2$

Step 4: Corrections

On output qubit 4:

$$X^{s_2} Z^{s_1 \oplus s_3}$$

On output qubit 6:

$$X^{s_3} Z^{s_5}$$

Complete pattern:

```
# Preparation
N_1 N_2 N_3 N_4 N_5 N_6      # Create 6 qubits in |+>
E_12 E_23 E_34 E_35 E_56    # Create cluster edges

# Measurements (in order)
M_1^0 → s_1                  # H gate
M_5^0 → s_5                  # CNOT target input
[M_2^0]^{s_1} → s_2          # Adapted by s_1
[M_3^{\pi/4}]^{s_2} → s_3    # S gate, adapted by s_2

# Corrections
X_4^{s_2} Z_4^{s_1 \oplus s_3} # Output qubit 1
X_6^{s_3} Z_6^{s_5}          # Output qubit 2
```

Output: Qubits 4 and 6 contain the circuit output state.

[← Back to Exercise 6.7](#)