Quantum Computing

Chapter 04: Computation, Measurement, and Dynamic Circuits

Prachya Boonkwan

Version: January 14, 2026

Sirindhorn International Institute of Technology
Thammasat University, Thailand

Who? Me?

- Nickname: Arm (P'/N' Arm, etc.)
- Born: Aug 1981
- Work
  - Researcher at NECTEC 2005-2024
  - Lecturer at SIIT, Thammasat University 2025-now
- Education
  - B.Eng & M.Eng in Computer Engineering, Kasetsart University, Thailand
  - Obtained Ministry of Science and Technology Scholarship of Thailand in early 2008
  - Did a PhD in Informatics (AI & Computational Linguistics) at University of Edinburgh, UK from 2008 to 2013

Table of Contents

Revision

Clifford Set of Quantum Gates

Gates	Symbols	From $\|0\rangle$	From $\|1\rangle$
NOT	$\mathbf{X}$	$\|1\rangle$	$\|0\rangle$
Hadamard	$\mathbf{H}$	$\frac{\|0\rangle+\|1\rangle}{\sqrt{2}}$	$\frac{\|0\rangle-\|1\rangle}{\sqrt{2}}$
Phase shift	$\mathbf{S}$	$\|0\rangle$	$i\|1\rangle$
Pauli-X	$\mathbf{X}$	$\|1\rangle$	$\|0\rangle$
Pauli-Y	$\mathbf{Y}$	$i\|1\rangle$	$-i\|0\rangle$
Pauli-Z	$\mathbf{Z}$	$\|0\rangle$	$-\|1\rangle$

Table 1: Single-qubit gates

Gates	Symbols	Descriptions
CNOT	$\mathbf{CNOT}$	$\|0,x\rangle \mapsto \|0,x\rangle$
		$\|1,x\rangle \mapsto \|1,\overline{x}\rangle$
Swap	$\mathbf{SWAP}$	$\|x,y\rangle \mapsto \|y,x\rangle$
CZ	$\mathbf{CZ}$	$\|0,x\rangle \mapsto \|0,x\rangle$
		$\|1,x\rangle \mapsto \|1,\mathbf{Z}\,x\rangle$

Table 2: Two-qubit gates

Operator Augmentation

- If we have a quantum operator

$$\mathbf{Q} = \sum_{k=1}^{K} |\phi_k\rangle \langle \psi_k|$$

we can augment $\mathbf{Q}$ with trivial qubits by

$$\mathbf{I}^M \otimes \mathbf{Q} = \sum_{k=1}^{K} |x_1, \dots, x_M, \phi_k\rangle \langle x_1, \dots, x_M, \psi_k|$$

$$\mathbf{Q} \otimes \mathbf{I}^M = \sum_{k=1}^{K} |\phi_k, x_1, \dots, x_M\rangle \langle \psi_k, x_1, \dots, x_M|$$

- This is useful for accommodating all entangled qubits, some of which not involved with the operator

- For any quantum operator $\mathbf{Q}$, we can augment it with trivial qubits by

$$\mathbf{Q}^{(N)}_{p_1,\dots,p_M} = \sum_{k=1}^{K} \left| x'_1, \dots, x'_N \right\rangle \langle x_1, \dots, x_N|$$

where the output mixed state is specified by the indices $p_1, \dots, p_M$, i.e.

$$\left| x'_{p_1}, \dots, x'_{p_M} \right\rangle = \mathbf{Q} \left| x_{p_1}, \dots, x_{p_M} \right\rangle$$

while the remaining input qubits are unchanged, i.e. $x'_{p_j} = x_{p_j}$ for all indices $j \notin \{p_1, \dots, p_M\}$

Non-Clifford Set

Non-Clifford Set of Quantum Logic Gates

- Limitation: We cannot construct all arbitrary quantum operators using only Clifford gates; for example,

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & \operatorname{cis}\frac{\pi}{4} \end{bmatrix}$$

- Gottesman-Knill Theorem: "Some quantum algorithms can be efficiently simulated by classical computers, even though they use superposition and entanglement."

- Non-Clifford Set: All quantum operators that cannot be constructed using only the Clifford gates
- Additional building blocks are needed: Toffoli gate, phase shift gate, rotation gates, and universal gate
- Simulation of these gates on classical computers is inefficient, which means it takes non-polynomial computation time
- When combined with superpositions and entanglement, these gates make some NP-hard problems solvable in polynomial time complexity

Toffoli Gate

- Toffoli gate flips the third qubit only if the first and second qubits are $|1\rangle$

$$|00x\rangle \mapsto |00x\rangle \qquad |01x\rangle \mapsto |01x\rangle$$
$$|10x\rangle \mapsto |10x\rangle \qquad |11x\rangle \mapsto |11\overline{x}\rangle$$

- The quantum operator $\mathbf{CCX}$ is

$$\mathbf{CCX} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- Example:

$$\mathbf{CCX}\left(\frac{|000\rangle + |111\rangle}{\sqrt{2}}\right) = \frac{|000\rangle + |110\rangle}{\sqrt{2}}$$

- $\mathbf{CCX}$ is self-inverse
- `qml.Toffoli(wires=…)`

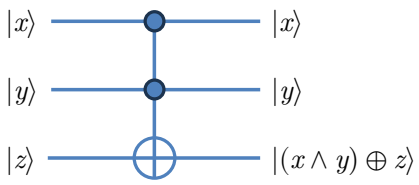


Figure 1: Toffoli gate

Boolean Universality of Toffoli Gate

- Toffoli gate is said to be universal: we can construct any classical Boolean logic gates from it, similar to NAND gate
- Theoretically, it enables quantum computers to emulate the classical ones with an FPGA-like Toffoli circuit

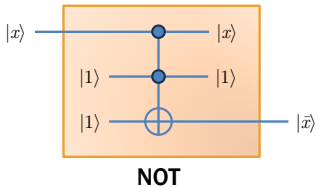


Figure 2: Classical NOT emulated with a Toffoli gate

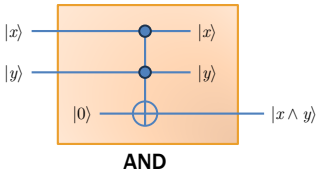


Figure 3: Classical AND emulated with a Toffoli gate

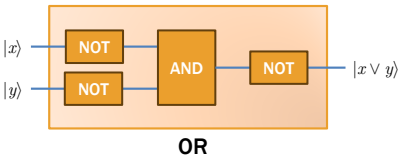


Figure 4: Classical OR emulated with a Toffoli gate

Generalized Controlled Gate

- Any single-qubit quantum operator $\mathbf{U}$ can be expressed in the exponential form

$$\mathbf{U} = \text{cis}(\mathbf{H}_\mathbf{U})$$

where $\mathbf{H}_\mathbf{U}$ is the Hermitian matrix of $\mathbf{U}$

- Generalized controlled gate of $\mathbf{U}$ is denoted by $\mathbf{CU}$, where

$$\mathbf{CU} = \text{cis}\left(\frac{(\mathbf{I}-\mathbf{Z})\otimes\mathbf{H}_\mathbf{U}}{2}\right) = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & u_{11} & u_{12} \\ 0 & 0 & u_{21} & u_{22} \end{array}\right]$$

- `qml.ctrl(opr, control=…)`
- Example: $\mathbf{CH} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$

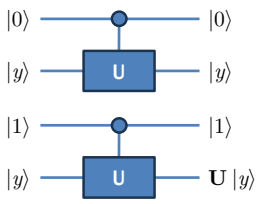


Figure 5: Controlled gate

Example 1

```
import pennylane as qml
from pennylane import numpy as np

dev = qml.device('default.qubit', wires=[0,1,2,3,4,5])

@qml.qnode(dev)
def circuit():
    # Prepare wires[0,1] = |00> + |11>
    qml.H(wires=[0])
    qml.CNOT(wires=[0,1])

    # Preapre wires[2,3] = |00> + i|11>
    qml.H(wires=[2])
    qml.ctrl(qml.Y(wires=[3]), control=[2])

    # Compute wires[0,1] AND wires[2,3]
    qml.Toffoli(wires=[0,2,4])
    qml.Toffoli(wires=[1,3,5])

    # Expect |000000> + |001100> + i|110000> + i|111111>
    return qml.state()

print(circuit())
```

Exercise 4.1: Toffoli Gate and Controlled Gates

Answer the following questions.

1. Design a quantum algorithm emulating an XOR gate using only Toffoli gates.
2. Design a quantum algorithm emulating a NAND gate using only Toffoli gates.
3. Design a quantum algorithm emulating a NOR gate using only Toffoli gates.
4. Why do we have to emulate classical logic gates using Toffoli gate? What are the properties superior to the classical Boolean algebra?
5. Explain why simulation of a Toffoli gate is inefficient on classical computers.
6. Design a quantum algorithm that imitates the gate $\mathbf{CU}$, where $\mathbf{U}$ is a unitary matrix, using only the Toffoli gate and the gate $\mathbf{U}$.

Design quantum algorithms that compute the following quantum-logical expressions. Develop PennyLane code to verify them.

7. $|11\rangle \wedge |10\rangle$
8. $|01\rangle \vee |10\rangle$
9. $\overline{|010\rangle}$
10. $\frac{|00\rangle+|11\rangle}{\sqrt{2}} \wedge |10\rangle$
11. $\frac{|00\rangle+|11\rangle}{\sqrt{2}} \oplus \frac{|01\rangle+|10\rangle}{\sqrt{2}}$
12. $\frac{|00\rangle+|11\rangle}{\sqrt{2}} \vee |01\rangle$

Generalized Phase Shift Gate

- We can generalize the phase shift gate so that rotation can be done more freely

$$\mathbf{P}(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & \operatorname{cis}\theta \end{bmatrix}$$

- `qml.PhaseShift(angle, wires=…)`
- Some Clifford gates can be redefined in terms of the generalized phase shift gate

$$\mathbf{Z} = \mathbf{P}(\pi) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\mathbf{S} = \mathbf{P}(\pi/2) = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

- Let's introduce the T gate

$$\mathbf{T} = \mathbf{P}(\pi/4) = \begin{bmatrix} 1 & 0 \\ 0 & \operatorname{cis}\frac{\pi}{4} \end{bmatrix}$$

- `qml.T(wires=…)`
- Note that $\mathbf{S} = \mathbf{Z}^{1/2}$ and $\mathbf{T} = \mathbf{Z}^{1/4}$
- $\mathbf{P}(\theta)$ is not always Hermitian, so we have to incorporate these adjoint gates to the instruction set

$$\mathbf{S}^* = \mathbf{P}(-\pi/2)$$
$$\mathbf{T}^* = \mathbf{P}(-\pi/4)$$

- `qml.adjoint(opr)`

Generalized Rotation Gates

- Let $\mathbf{M}$ be a unitary matrix (i.e. Hermitian $\mathbf{M} = \mathbf{M}^*$ and orthonormal $\mathbf{M}^{-1} = \mathbf{M}^*$), where

$$\mathbf{M} = \begin{bmatrix} \alpha & \gamma \\ \beta & \delta \end{bmatrix}$$

- Let's define a generalized rotation gate for a unitary matrix $\mathbf{M}$ as follows

$$\begin{aligned} \mathbf{R}_\mathbf{M}(\theta) &= \text{cis}\left(-\frac{i\theta}{2}\mathbf{M}\right) \\ &= \cos\left(\frac{\theta}{2}\right)\mathbf{I} - i\sin\left(\frac{\theta}{2}\right)\mathbf{M} \end{aligned}$$

- General form:

$$\mathbf{R}_\mathbf{M}(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} - i\alpha\sin\frac{\theta}{2} & -i\gamma\sin\frac{\theta}{2} \\ -i\beta\sin\frac{\theta}{2} & \cos\frac{\theta}{2} - i\delta\sin\frac{\theta}{2} \end{bmatrix}$$

- Mapping:

$$|0\rangle \mapsto \left[\cos\frac{\theta}{2} - i\alpha\sin\frac{\theta}{2}\right]|0\rangle - \left[i\beta\sin\frac{\theta}{2}\right]|1\rangle$$

$$|1\rangle \mapsto \left[-i\gamma\sin\frac{\theta}{2}\right]|0\rangle + \left[\cos\frac{\theta}{2} - i\delta\sin\frac{\theta}{2}\right]|1\rangle$$

X-Rotation Gates

- X-Rotation gate flips the qubit

$$\begin{aligned} \mathrm{RX}(\theta) &= \mathrm{cis}\left(-\frac{i\theta}{2}\mathrm{X}\right) \\ &= \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \end{aligned}$$

- Mapping:

$$\begin{aligned} |0\rangle &\mapsto \cos\frac{\theta}{2}|0\rangle - i\sin\frac{\theta}{2}|1\rangle \\ |1\rangle &\mapsto -i\sin\frac{\theta}{2}|0\rangle + \cos\frac{\theta}{2}|1\rangle \end{aligned}$$

- `qml.RX(angle, wires=…)`

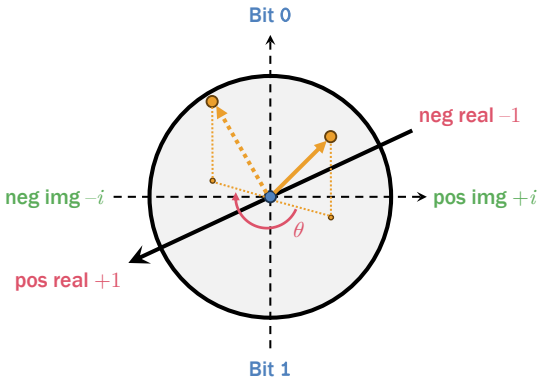


Figure 6: X-Rotation gate

Y-Rotation Gates

- Y-Rotation gate flips the qubit, the sign, and the amplitude

$$\begin{aligned} \mathbf{RY}(\theta) &= \operatorname{cis}\left(-\frac{i\theta}{2}\mathbf{Y}\right) \\ &= \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \end{aligned}$$

- Mapping:

$$\begin{aligned} |0\rangle &\mapsto \cos\frac{\theta}{2}|0\rangle + \sin\frac{\theta}{2}|1\rangle \\ |1\rangle &\mapsto -\sin\frac{\theta}{2}|0\rangle + \cos\frac{\theta}{2}|1\rangle \end{aligned}$$

- `qml.RY(angle, wires=…)`

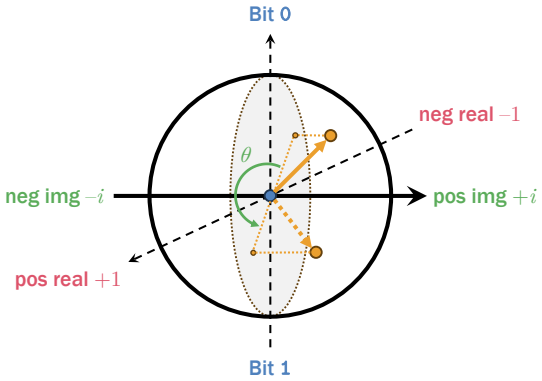


Figure 7: Y-Rotation gate

Z-Rotation Gates

- Z-Rotation gate flips the sign

$$\begin{aligned} \mathrm{RZ}(\theta) &= \mathrm{cis}\left(-\frac{i\theta}{2}\mathrm{Z}\right) \\ &= \begin{bmatrix} \mathrm{cis}\left(-\frac{\theta}{2}\right) & 0 \\ 0 & \mathrm{cis}\,\frac{\theta}{2} \end{bmatrix} \end{aligned}$$

- Mapping:

$$\begin{aligned} |0\rangle &\mapsto \cos\left(-\frac{\theta}{2}\right)|0\rangle \\ |1\rangle &\mapsto \cos\frac{\theta}{2}|1\rangle \end{aligned}$$

- `qml.RZ(angle, wires=…)`

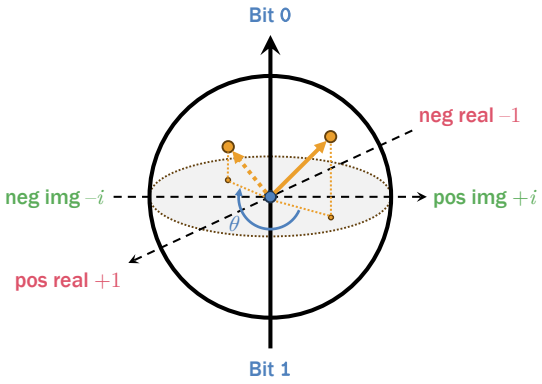
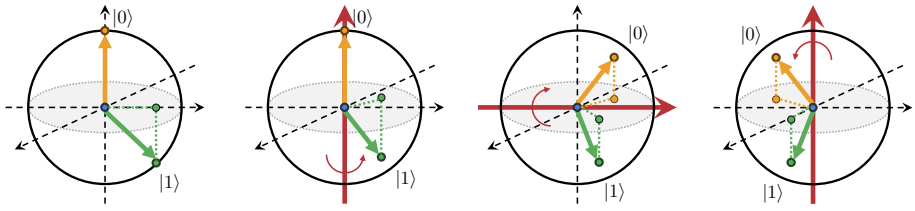


Figure 8: Z-Rotation gate

Universality of Phase Shift and Rotation Gates

- Any single-qubit operator $\mathbf{U}$ can be simulated by: $\mathbf{U} = \mathbf{RZ}(\phi)\ \mathbf{RX}(\theta)\ \mathbf{RZ}(\omega)\ \mathbf{P}(\lambda)$
- Equivalently, we can also replace $\mathbf{RX}(\theta)$ with $\mathbf{RY}(\theta)$ to flip qubits, signs, and amplitudes
- Universal quantum gate set: $\mathbf{P}$, $\mathbf{RX}$, $\mathbf{RY}$, $\mathbf{RZ}$, and $\mathbf{CNOT}$



Step 0:
Shift the phase of $|1\rangle$ by λ

Step 1:
Rotate $|0\rangle$ and $|1\rangle$ around Axis Z by phase $\omega/2$

Step 2:
Rotate $|0\rangle$ and $|1\rangle$ around Axis X by phase $\theta/2$

Step 3:
Rotate $|0\rangle$ and $|1\rangle$ around Axis Z by phase $\phi/2$

Figure 9: Generalized rotation gates

Exercise 4.2: Generalized Phase Shift and Rotation Gates

Compute quantum operators **P**, **RX**, **RY**, and **RZ** of the following phases.

1. 0
2. $\pi/6$
3. $\pi/4$
4. $\pi/3$
5. $\pi/2$
6. $2\pi/3$
7. $3\pi/4$
8. $5\pi/6$
9. π
10. 2π

Design quantum algorithms that prepare the following states using generalized phase shift and rotation gates. Develop PennyLane code to verify each of them.

11. $|0\rangle$
12. $|1\rangle$
13. $-|1\rangle$
14. $|+\rangle$
15. $|-\rangle$
16. $\text{cis}\,\frac{\pi}{4}\,|0\rangle + \text{cis}\,\frac{\pi}{8}\,|1\rangle$
17. $\text{cis}\,\frac{\pi}{2}\,|0\rangle - \text{cis}\,\frac{\pi}{2}\,|1\rangle$
18. $\text{cis}\,\frac{\pi}{3}\,|0\rangle + |1\rangle$

Measurement

State Measurement

- Observing the final quantum state of a quantum algorithm
- Type 1: Analytical solution
- Type 2: Projective measurement
- Type 3: State expectation
- Type 4: Variance
- Type 5: Simulation

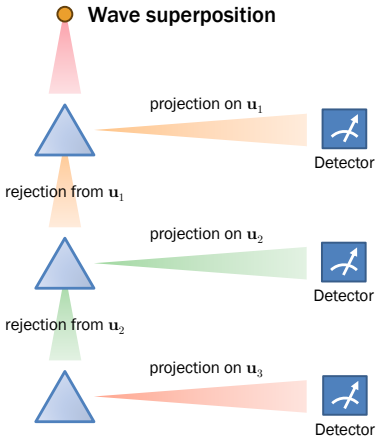


Figure 10: Projective measurement

Type 1: Analytical Solution

- To observe the final state theoretically by emulating a quantum algorithm
- PennyLane: `qml.state()`
- In reality, measured quantum states may deviate from the analytical solution due to quantum decoherence (i.e. noise)
- Any quantum algorithm has to be evaluated by
  - Computing the analytical solution; this is the expected output
  - Running on an actual quantum computer to measure noise sensitivity of the algorithm

Figure 11: Quantum computer

Example 2 (Analytical Solution)

```
import pennylane as qml
from pennylane import numpy as np

dev = qml.device('default.qubit', wires=[0, 1])

@qml.qnode(dev)
def circuit(theta: float):

    qml.RY(theta, wires=[0])
    qml.CNOT(wires=[0,1])
    qml.RZ(theta, wires=[1])

    # Analytical solution
    return qml.state()

print(circuit(np.pi / 3))
```

Expected output: $\begin{bmatrix} \frac{3-\sqrt{3}i}{4} & 0 & 0 & \frac{\sqrt{3}+i}{4} \end{bmatrix}^\top$

```
[0.75-0.433j    0.    0.    0.433+0.25j]
```

Type 2: Projective Measurement

- To observe a quantum state with laser-beam projection
- We can measure a qubit $|\psi\rangle = a|0\rangle + b|1\rangle$ by collapsing it, i.e. projecting it on a set of unitary vectors $\mathbf{O} = \{|u_1\rangle, \dots, |u_N\rangle\}$

$$\begin{aligned} \text{proj}_{|u_k\rangle}|\psi\rangle &= \frac{\langle\psi|u_k\rangle}{\langle u_k|u_k\rangle}|u_k\rangle \\ &= \left(|\psi\rangle^\top \overline{|u_k\rangle}\right)|u_k\rangle \\ &= \left(a\bar{c}_k + b\bar{d}_k\right)|u_k\rangle \end{aligned}$$

where each $|u_k\rangle = c_k|0\rangle + d_k|1\rangle$ is called an observable

- In measurement, the qubit is destroyed; i.e. measurement is irreversible

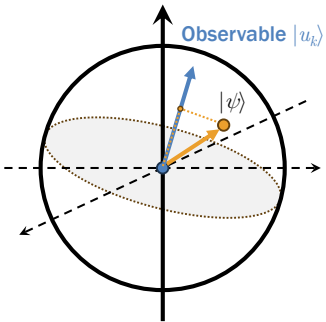


Figure 12: Projective measurement

Information Extraction

- When decomposed into observables $|u_k\rangle = c_k|0\rangle + d_k|1\rangle$, we can measure the probability for each of them as follows

$$P(u_k|\psi) = \left|a\bar{c}_k + b\bar{d}_k\right|^2$$

- Information extraction: We collapse a qubit on pure states $\mathbf{U} = \{|0\rangle, |1\rangle\}$

$$\begin{aligned} P(0|\psi) &= |a|^2 \\ P(1|\psi) &= |b|^2 \end{aligned}$$

  This is a distribution of binary values

- PennyLane: `qml.probs()`

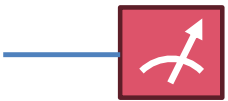


Figure 13: Measurement gate

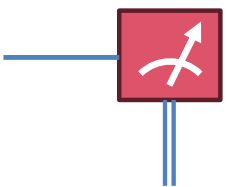


Figure 14: Measurement gate with control output

Example 2 (Projective Measurement)

```
import pennylane as qml
from pennylane import numpy as np

dev = qml.device('default.qubit', wires=[0, 1])

@qml.qnode(dev)
def circuit(theta: float):

    qml.RY(theta, wires=[0])
    qml.CNOT(wires=[0,1])
    qml.RZ(theta, wires=[1])

    # Projective measurement
    return qml.probs()

print(circuit(np.pi / 3))
```

Expected output: $\begin{bmatrix} 0.75 & 0 & 0 & 0.25 \end{bmatrix}^\top$

```
[0.75    0.    0.    0.25]
```

Information Extraction via Quantum Operator

- From Chapter 1, any quantum operator $\mathbf{U}$ can be eigen-decomposed into

$$\mathbf{U} = \sum_{k=1}^{N} \lambda_k |u_k\rangle \langle u_k|$$

where $\{(\lambda_k, |u_k\rangle)\}_{k=1}^{N} = \text{eigen}(\mathbf{U})$

- Pauli-Z Gate

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

Its eigen-pairs are $(1, |0\rangle)$ and $(-1, |1\rangle)$

- $|0\rangle$ and $|1\rangle$ are called Z-bases

- Pauli-X Gate

$$\mathbf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Its eigen-pairs are $(1, |+\rangle)$ and $(-1, |-\rangle)$

- $|+\rangle$ and $|-\rangle$ are called X-bases

- Pauli-Y Gate

$$\mathbf{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Its eigen-pairs are $(1, |+i\rangle)$ and $(-1, |-i\rangle)$

- $|+i\rangle$ and $|-i\rangle$ are called Y-bases

Type 3: Multi-Shot Measurement with Expectation

- Since quantum computing is probabilistic, we sometimes repeat an experiment to obtain the distribution of outcomes
- The expectation of quantum operator $\mathbf{U}$ on a state $|\psi\rangle$ is

$$\begin{aligned}\left\langle \mathbf{U}^{(\psi)} \right\rangle &= \langle\psi|\ \mathbf{U}\ |\psi\rangle \\ &= \sum_{k=1}^{K} \lambda_k \left|\langle u_k|\psi\rangle\right|^2\end{aligned}$$

where $(\lambda_k, |u_k\rangle)$ is the k-th eigen-pair of $\mathbf{U}$

- In actual experiments, each state is a binary string sampled from $|\psi\rangle$

- The expectation is approximated by

$$\begin{aligned}\left\langle \mathbf{U}^{(\psi)} \right\rangle &= \mathbb{E}_{|s_j\rangle\sim|\psi\rangle}\left[\sum_{k=1}^{K} \lambda_k \left|\langle u_k|s_j\rangle\right|^2\right] \\ &\approx \frac{1}{N}\sum_{j=1}^{N}\sum_{k=1}^{K} \lambda_k \left|\langle u_k|s_j\rangle\right|^2\end{aligned}$$

where N is an adequately large number

Figure 15: Measurement of quantum operator $\mathbf{U}$

Expectation on Pauli Projectors

- Suppose $|\psi\rangle = a|0\rangle + b|1\rangle$
- Pauli-Z Projector: Positive value implies more chance of pointing to the $|0\rangle$ hemisphere

$$\begin{aligned}\left\langle \mathbf{Z}^{(\psi)} \right\rangle &= 1\langle 0|\psi\rangle - 1\langle 1|\psi\rangle \\ &= |a|^2 - |b|^2\end{aligned}$$

- Pauli-X Projector: Positive value implies more chance of pointing to the positive real hemisphere

$$\begin{aligned}\left\langle \mathbf{X}^{(\psi)} \right\rangle &= 1\langle +|\psi\rangle - 1\langle -|\psi\rangle \\ &= |a+b|^2 - |a-b|^2\end{aligned}$$

- Pauli-Y Projector: Positive value implies more chance of pointing to the positive imaginary hemisphere

$$\begin{aligned}\left\langle \mathbf{Y}^{(\psi)} \right\rangle &= 1\langle +i|\psi\rangle - 1\langle -i|\psi\rangle \\ &= |a+bi|^2 - |a-bi|^2\end{aligned}$$

Gates	Positive	Negative
Pauli-Z	$\|0\rangle$	$\|1\rangle$
Pauli-X	Pos. real	Neg. real
Pauli-Y	Pos. img	Neg. img

Table 3: Implication of expectation

Joint Expectation as Non-Destructive Projection

- Projection onto a single axis is irreversible, because it destroys, or decoheres, the quantum state throughout the complex vector space
- However, joint projection on multiple axes is reversible, because it makes projection onto a subspace; therefore, it does not decohere the quantum state
- For example, there are four possible axes for a two-qubit system: $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$, and we choose to project a quantum state onto only two eigen-states $|00\rangle$ and $|11\rangle$

- Joint projection on two states $|\psi_1\rangle$ on projector $\mathbf{U}_1$ and $|\psi_2\rangle$ on $\mathbf{U}_2$ is

$$P(\psi_1, \psi_2 | \mathbf{U}_1, \mathbf{U}_2) = \left\langle \mathbf{U}_1^{(\psi_1)} \right\rangle \left\langle \mathbf{U}_2^{(\psi_2)} \right\rangle$$

- We can compute the joint probability, a.k.a. joint projection, for residing in each positive side

$$P(\psi_1, \dots, \psi_N | \mathbf{U}_1, \dots, \mathbf{U}_N) = \prod_{k=1}^{N} \left\langle \mathbf{U}_k^{(\psi_k)} \right\rangle$$

where each state $|\psi_k\rangle$ is measured on the quantum operator $\mathbf{U}_k$

Mixing and Matching of Expectations

- PennyLane:
  - `qml.expval(op=qml.Z(wires=…))`
  - `qml.expval(op=qml.X(wires=…))`
  - `qml.expval(op=qml.Y(wires=…))`
- PennyLane:

```
return qml.expval(
    qml.Op1(wires=...)
    @ qml.Op2(wires=...)
    @ qml.Op3(wires=...)
    ...
    @ qml.OpN(wires=...)
)
```

where each $\texttt{qml.Op}_k$ is one of the projectors X, Y, and Z

- Linear combination of projectors:

```
return qml.expval(
    0.5 * qml.X(wires=[0])
    + 0.1 * qml.Y(wires=[1])
    + 0.4 * qml.Z(wires=[2])
)
```

Also: `qml.ops.LinearCombination(coeffs, projectors)`

- Returning several expectations is also possible in PennyLane

```
return (
    qml.expval(qml.Op1(wires=...)),
    qml.expval(qml.Op2(wires=...)),
    qml.expval(qml.Op3(wires=...)),
    .
    .
)
```

Type 4: Variance

- In multi-shot measurement, it is useful to measure the disperse in an outcome distribution
- The variance of quantum operator $\mathbf{U}$ on a state $|\psi\rangle$ is

$$\begin{aligned} \mathrm{Var}\left[\mathbf{U}^{(\psi)}\right] &= \left\langle \left(\mathbf{U}^2\right)^{(\psi)} \right\rangle - \left\langle \mathbf{U}^{(\psi)} \right\rangle^2 \\ &= \langle\psi|\mathbf{U}^2|\psi\rangle - \langle\psi|\mathbf{U}|\psi\rangle^2 \end{aligned}$$

- PennyLane:
  - `qml.var(op=qml.Z(wires=…))`
  - `qml.var(op=qml.X(wires=…))`
  - `qml.var(op=qml.Y(wires=…))`

- With the expectation and variance on the same quantum operator $\mathbf{U}$, we can altogether compare any two given qubits by addressing two questions
  - Which one goes to which extreme?
  - Which one has a more consistent or volatile distribution?
- When measuring a qubit with several quantum operators, the one with the lowest variance is the most consistent

Example 2 (Expectation & Variance)

```
import pennylane as qml
from pennylane import numpy as np

dev = qml.device('default.qubit', wires=[0, 1])

@qml.qnode(dev)
def circuit(theta: float):

    qml.RY(theta, wires=[0])
    qml.CNOT(wires=[0,1])
    qml.RZ(theta, wires=[1])

    # Expectation of each wire
    return ( qml.expval(qml.Z(wires=[0])), qml.expval(qml.Z(wires=[1])),
             qml.var(qml.Z(wires=[0])), qml.var(qml.Z(wires=[1])) )

print(circuit(np.pi / 3))
```

Expected output: (0.5, 0.5, 0.75, 0.75)

```
(0.5, 0.5, 0.75, 0.75)
```

Type 5: Simulation

- Simulation is based on sampling mixed states from their probability distribution
- For a superposition

$$|\psi\rangle \quad = \quad \alpha_1 |\psi_1\rangle + ... + \alpha_K |\psi_K\rangle$$

The probability distribution for each mixed state $|\psi_k\rangle$ is

$$P(\psi_k|\psi) \quad = \quad |\alpha_k|^2$$

- We can then sample a mixed state by

$$\left|\psi^{(j)}\right\rangle \quad \sim \quad P(\cdot|\psi)$$

- Suppose we have two superpositions

$$\begin{aligned} |\psi\rangle &= \alpha_1 |\psi_1\rangle + ... + \alpha_K |\psi_K\rangle \\ |\phi\rangle &= \beta_1 |\phi_1\rangle + ... + \beta_M |\phi_M\rangle \end{aligned}$$

The probability distribution of their tensor product $|\psi\rangle \otimes |\phi\rangle$ is

$$P(\psi_j \phi_k|\psi, \phi) \quad = \quad \left|\alpha_j \beta_k\right|^2$$

- We can sample a tensor product by

$$\left|\psi_j \phi_k\right\rangle \quad \sim \quad P(\cdot|\psi, \phi)$$

Simulation Methods

- By specifying the option `shots` in the instruction `qml.device`, we enable the simulation mode
- Two simulation methods: step-by-step and end-to-end
- Step-by-step sampling: draw a set of N bitstrings from the probability distribution repeatedly
  - `shots` = the number of shots for step-by-step sampling
  - PennyLane: `qml.sample()`
  - Output: a list of qubit sequences
- End-to-end simulation: draw N bitstrings from the probability distribution only once
  - `shots` = the number of shots for step-by-step sampling
  - PennyLane: `qml.counts()`
  - Output: a frequency table for each possible bitstring

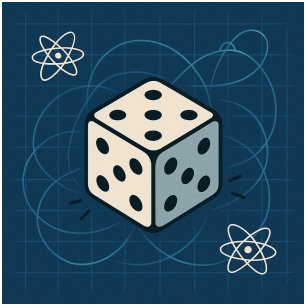


Figure 16: Quantum dice roll

Example 2 (Sampling)

```
import pennylane as qml
from pennylane import numpy as np

# Step-by-step simulation by 10-shots sampling
dev = qml.device('default.qubit', wires=[0, 1], shots=10)    # shots = number of samples

@qml.qnode(dev)
def circuit(theta: float):

    qml.RY(theta, wires=[0])
    qml.CNOT(wires=[0,1])
    qml.RZ(theta, wires=[1])

    # Sampling
    return qml.sample()

print(circuit(np.pi / 3))
```

Result: $\sim 75\%$ of $|00\rangle$ and $\sim 25\%$ of $|11\rangle$

```
[ [0 0], [0 0], [0 0], [0 0], [1 1] ]
```

Example 2 (End-to-End Simulation)

```
import pennylane as qml
from pennylane import numpy as np

# End-to-end simulation with 1000 shots
dev = qml.device('default.qubit', wires=[0, 1], shots=1000)    # shots = number of samples

@qml.qnode(dev)
def circuit(theta: float):

    qml.RY(theta, wires=[0])
    qml.CNOT(wires=[0,1])
    qml.RZ(theta, wires=[1])

    # Simulation
    return qml.counts()

print(circuit(np.pi / 3))
```

Result: $\sim 75\%$ of $|00\rangle$ and $\sim 25\%$ of $|11\rangle$

```
{'00': 738, '11': 262}
```

Density Matrix

- Density matrix: a unified form of pure states and mixed states
- A density matrix for $|\psi\rangle = a|0\rangle + b|1\rangle$ is

$$\mathbf{R}^{(\psi)} = \begin{bmatrix} |a|^2 & ab^* \\ a^*b & |b|^2 \end{bmatrix}$$

  - The diagonal entries are the probability of $|0\rangle$ and $|1\rangle$, respectively
  - The off-diagonal entries are phase coherence
- Trace: sum of the diagonal line

$$\mathrm{tr}(\mathbf{A}) = \sum_{k=1}^{N} A_{k,k}$$

- Property: Quantum state $|\psi\rangle$ is pure if and only if $\mathrm{tr}\left(\left[\mathbf{R}^{(\psi)}\right]^2\right) = 0$
- Expectation of quantum operator $\mathbf{U}$ on a state $|\psi\rangle$ can be rewritten as

$$\left\langle \mathbf{U}^{(\psi)} \right\rangle = \mathrm{tr}(\mathbf{R}^{(\psi)}\mathbf{U})$$

- Example:

$$\begin{aligned} \langle \mathbf{Z}^{\psi} \rangle &= \mathrm{tr}(\mathbf{R}^{(\psi)}\mathbf{Z}) \\ &= \mathrm{tr}\left(\begin{bmatrix} |a|^2 & ab^* \\ a^*b & |b|^2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \right) \\ &= |a|^2 - |b|^2 \end{aligned}$$

Reconstructing a Pure State from a Density Matrix

1. With the expectations of all Pauli projectors, we can reconstruct a qubit $|\psi\rangle$ in terms of a density matrix

$$\begin{aligned}\mathbf{R} &= \frac{1}{2}\left(\mathbf{I} + \langle \mathbf{X}\rangle \mathbf{X} + \langle \mathbf{Y}\rangle \mathbf{Y} + \langle \mathbf{Z}\rangle \mathbf{Z}\right) \\ &= \frac{1}{2}\begin{bmatrix} 1+\langle \mathbf{Z}\rangle & \langle \mathbf{X}\rangle - i\langle \mathbf{Y}\rangle \\ \langle \mathbf{X}\rangle + i\langle \mathbf{Y}\rangle & 1-\langle \mathbf{Z}\rangle \end{bmatrix}\end{aligned}$$

2. Check if $\mathbf{tr}(\mathbf{R}^2) = 0$. If so, the qubit will be pure and we can proceed.
   2.1 We eigen-decompose $\mathbf{R}$ into eigen-pairs

$$\left\{(\lambda_k, |\psi_k\rangle)\right\}_{k=1}^{N} = \mathbf{eigen}(\mathbf{R})$$

   2.2 If any $\lambda_k = 1$, its eigenvector $|\psi_k\rangle$ becomes a state vector of $\mathbf{R}$.
3. Otherwise, it is a mixed state and the density matrix is already the best description for this qubit.

Exercise 4.3: Measurement and Density Matrix

Convert the following qubits into equivalent density matrices.

1. $|0\rangle$
2. $|1\rangle$
3. $i\,|0\rangle$
4. $-|1\rangle$
5. $\frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)$
6. $\frac{1}{5}\left(3\,|0\rangle + 4\,|1\rangle\right)$
7. $\frac{1}{\sqrt{2}}\left(|0\rangle + i\,|1\rangle\right)$
8. $\frac{1}{\sqrt{5}}\left(-i\,|0\rangle + 2\,|1\rangle\right)$
9. $\frac{1}{\sqrt{2}}\left(\operatorname{cis}\frac{\pi}{4}\,|0\rangle + \operatorname{cis}\frac{\pi}{3}\,|1\rangle\right)$
10. $\frac{4}{5}\operatorname{cis}\frac{2\pi}{3}\,|0\rangle + \frac{3}{5}\operatorname{cis}\frac{\pi}{3}\,|1\rangle$

Compute the projective measurements $\langle \mathbf{X} \rangle$, $\langle \mathbf{Y} \rangle$, and $\langle \mathbf{Z} \rangle$ from these density matrices, and check if they are pure states or mixed states.

11. $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$
12. $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$
13. $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$
14. $\begin{bmatrix} 1 & i \\ -i & 1 \end{bmatrix}$
15. $\frac{1}{2}\begin{bmatrix} 1.5 & 1-i \\ 1+i & 0.5 \end{bmatrix}$

Dynamic Circuit

Mid-Circuit Measurement

- We can introduce if-clauses and loops into a quantum algorithm, which means the quantum circuit dynamically changes w.r.t. the current state
- Mid-circuit measurement (MCM): A quantum state is measured in the middle of the algorithm, and optionally it can be reset
- Output: classical bit $\{0, 1\}$
- MCM is used in combination with the if-clause to form a conditional statement or a loop
- MCM can be done by taking a quick sneak peek of a qubit nondestructively, and estimating the actual quantum state — the result may be noisy

`qml.measure(wires=…)` Measure the classical bit of the specified wire.
`qml.measure(wires=…, reset=True)` Measure the classical bit of the specified wire and reset that wire to $\|0\rangle$.
`qml.measure(wires=…, postselect=`n`)` Measure the classical bit of the specified wire. If it is NOT equal to the postselected value, terminate this circuit run.

Table 4: Mid-circuit measurement

Conditional Operator

- We can perform quantum operators conditioned on the outcome of the measurement
- PennyLane: `qml.cond(bool_expr, opr)(params)`
- If the Boolean expression is true, then the operator is applied on the parameters
- We can form a complex Boolean expression by `&` (and), `|` (or), and `not` (negation)

`qml.cond(bool, true_fn=…)(params)`
If the condition is true, the `true_fn` is called with the specified parameters.

`qml.cond(bool, true_fn=…, false_fn=…)(params)`
If the condition is true, the `true_fn` is called with the specified parameters. Otherwise, the `false_fn` is called with the same parameters.

`qml.cond(bool, true_fn=…, false_fn=…, elifs=…)(params)`
The option `elifs` is a list of `(cond, opr)`, where each pair is a condition and its operation in the else-if part.

Condition Operator

- We can emulate the if-clause in quantum computing by treating the condition as a classical control bit

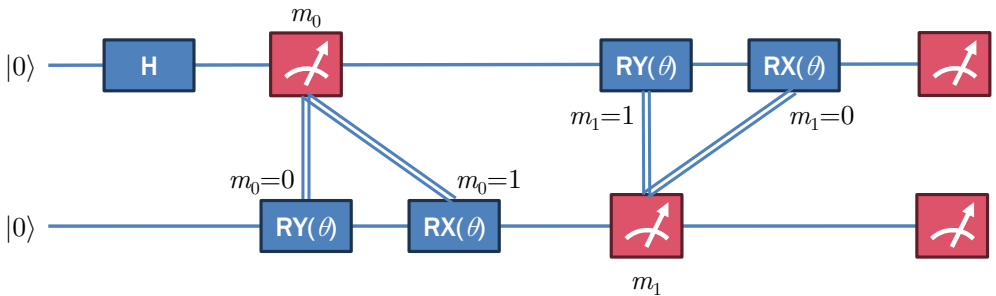


Figure 17: Quantum algorithm with mid-circuit measurement and conditional operators

Example 3: Conditional Operator

```
dev = qml.device('default.qubit', wires=[0, 1, 2], shots=500)

@qml.qnode(dev)
def circuit(theta: float):
    qml.Hadamard(wires=[0])         # wire-0 is |+>

    m0 = qml.measure(wires=[0])     # m0 is either 0 or 1
    qml.cond(m0 == 0,               # If m0 == 0:
        true_fn=qml.RY,             #     Y-rotate wire-1 by theta
        false_fn=qml.RX             # Else: X-rotate wire-1 by theta
    )(theta, wires=[1])             # (Common parameters)

    m1 = qml.measure(wires=[1])     # m1 is either 0 or 1
    qml.cond(m1 == 1,               # If m1 == 1
        true_fn=qml.RX,             #     X-rotate wire-2 by theta
        false_fn=qml.RZ             # Else: Z-rotate wire-2 by theta
    )(theta, wires=2)               # (Common parameters)

    return qml.counts()

# Result = {'000': 125, '001': 50, '010': 50, '100': 150, '101': 50, '110': 75}
# N.B. It sums to 500.
print(circuit(np.pi / 3))
```

Postselection

- Postselection: If the measurement is not equal to the specified value, we will discard this circuit run

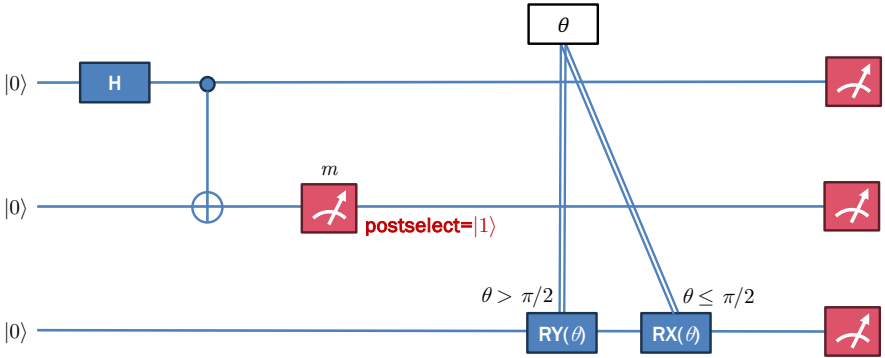


Figure 18: Quantum algorithm with postselection and condition on external parameter θ

Example 4: Postselection

```
dev = qml.device('default.qubit', wires=[0, 1, 2], shots=500)

@qml.qnode(dev)
def circuit(theta: float):

    qml.Hadamard(wires=[0])
    qml.CNOT(wires=[0, 1])         # Bell state |Psi+>
    qml.RX(theta, wires=[1])       # X-rotate wire-1 by theta

    # Postselection: If wire-1 is not 1, terminate this circuit run
    m = qml.measure(wires=[1], postselect=1)

    qml.cond(theta > np.pi / 2,    # If theta > pi / 2:
        true_fn=qml.RX,            #     X-rotate wire-2 by theta
        false_fn=qml.RY            # Else: Y-rotate wire-2 by theta
    )(theta, wires=[2])            # (Common parameters)

    return qml.counts()

# Result: {'010': 40, '011': 10, '110': 150, '111': 50}.
# N.B. It does not sum to 500 due to the postselection.
print(circuit(np.pi / 3))
```

Looping in PennyLane

- For-loop: Decorator `@qml.for_loop(start, stop, step)`
- While-loop: Decorator `@qml.while_loop(cond_fn)`
- The decorated function becomes the loop's body, where its parameters and returned values connect between each iteration
- Once you declare a function decorated with these decorators, you have to actually call it to run the loop — you are encouraged to modularize and parameterize your loops

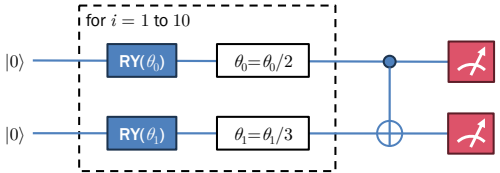


Figure 19: For-loop in quantum circuits

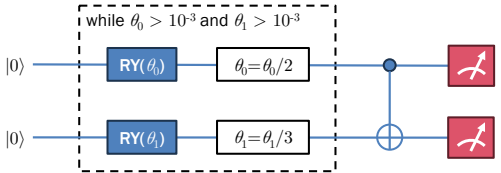


Figure 20: While-loop in quantum circuits

Example 5: For-Loop

```
dev = qml.device('default.qubit', wires=[0, 1], shots=500)

@qml.qnode(dev)
def circuit_for(iter_no: int, theta0: float, theta1: float):

    @qml.for_loop(0, iter_no, 1)          # for i in range(0, iter_no, 1):
    def loop(i, theta0, theta1):          #     (This is the loop's body)
        qml.RY(theta0, wires=[0])         #     Y-rotate wire-0 by theta0
        qml.RY(theta1, wires=[1])         #     X-rotate wire-1 by theta1
        theta0_new = theta0 / 2           #     Update theta0
        theta1_new = theta1 / 3           #     Update theta1
        return theta0_new, theta1_new     #     Pass theta0 and theta1 to the next iteration

    loop(theta0, theta1)                  # Actually run the loop
    qml.CNOT(wires=[0, 1])
    return qml.counts()

circuit_for(10, np.pi / 4, np.pi / 4)
```

Example 6: While-Loop

```
dev = qml.device('default.qubit', wires=[0, 1], shots=500)

@qml.qnode(dev)
def circuit_while(iter_no: int, theta0: float, theta1: float):

    @qml.while_loop(                    # while:
        lambda t0, t1:                  #     (Terminal condition)
            t0 > 1e-3 and t1 > 1e-3     #     theta0 > 1e-3 and theta1 > 1e-3
    )                                   #
    def loop(theta0, theta1):           #     (This is the loop's body)
        qml.RY(theta0, wires=[0])       #     Y-rotate wire-0 by theta0
        qml.RY(theta1, wires=[1])       #     Y-rotate wire-1 by theta1
        theta0_new = theta0 / 2         #     Update theta0
        theta1_new = theta1 / 3         #     Update theta1
        return theta0_new, theta1_new   #     Pass theta0 and theta1 to the next iteration

    loop(theta0, theta1)                # Actually run the loop
    qml.CNOT(wires=[0, 1])
    return qml.counts()

circuit_while(10, np.pi / 4, np.pi / 4)
```

Exercise 4.4: Dynamic Circuits

Develop PennyLane programs to compute the following states. Let each $|x_k\rangle$ be the k-th wire.

1. Compute $\left[\prod_{k=1}^{10} \mathbf{RX}(\theta/2^k)\right]|x_0\rangle$, for some $\theta > 0$.
2. Let

$$|y_0\rangle = \left[\prod_{k=1}^{5} \mathbf{RX}(\theta/2^k)\right]|x_0\rangle$$

$$|y_1\rangle = \left[\prod_{k=1}^{5} \mathbf{RZ}(\theta/3^k)\right]|x_1\rangle$$

   Compute $\mathbf{CNOT}|y_0, y_1\rangle$, for some $\theta > 0$.
3. Compute $\mathbf{CCX}|y_0, y_1, 0\rangle$, where each $|y_j\rangle = \prod_{k=1}^{5} \mathbf{RX}((2k+j)\theta)$, for some $\theta > 0$.
4. Compute $\mathbf{CCX}|y_0, y_1, 0\rangle$, where each $|y_j\rangle = \prod_{k=1}^{N} \mathbf{RX}(\theta/2^{2k+j})$, for some $\theta > 0$, and each phase $\theta/2^{2k+j} > 10^{-4}$. The number N varies depending on this condition.

Exercise 4.4 (cont'd): Dynamic Circuits

5. Compute the GHZ state as illustrated below.

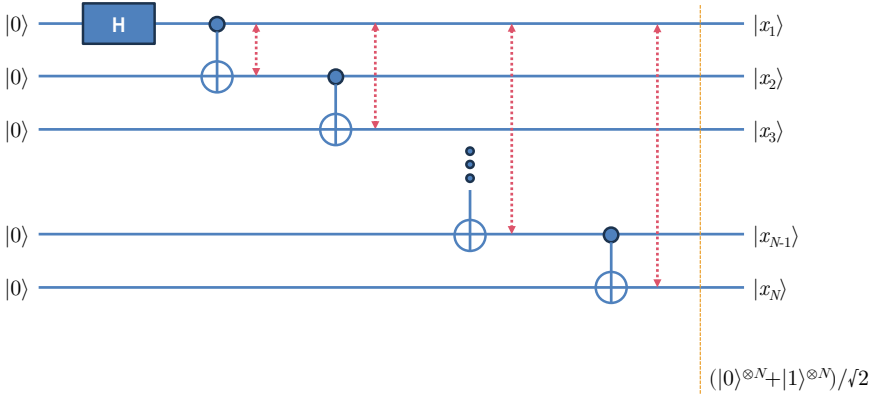


Figure 21: GHZ entangled state

Exercise 4.4 (cont'd): Dynamic Circuits

6. Compute the following quantum circuit. Each $\mathbf{R}_k$ is a phase shift gate with phase $2\pi/2^k$. [Reminder: Phase shift gate is `qml.PhaseShift(angle, wires=…)` in PennyLane]

$$\mathbf{R}_k = \mathbf{P}(2\pi/2^k)$$

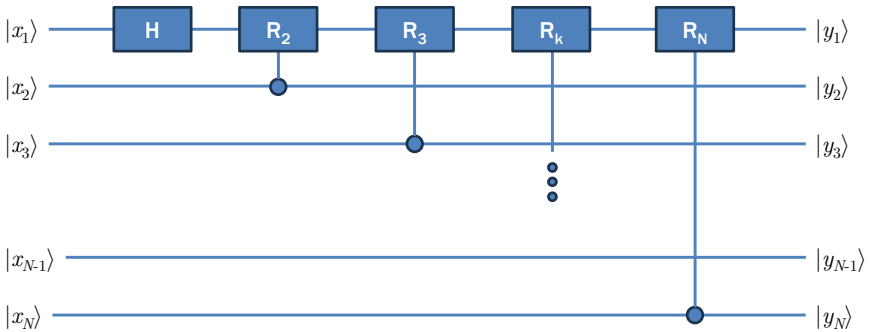


Figure 22: GHZ entangled state

Exercise 4.4 (cont'd): Dynamic Circuits

7. Compute the following circuit for quantum Fourier transform. Note that the computation for each $|y_k\rangle$ can be seen as a modularized loop in Question 6.

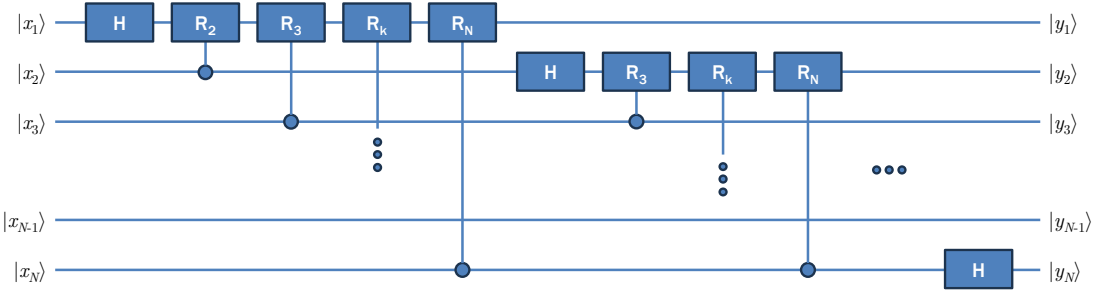


Figure 23: Quantum Fourier transform

Exercise 4.4 (cont'd): Dynamic Circuits

8. Compute the following circuit for quantum inverse Fourier transform.

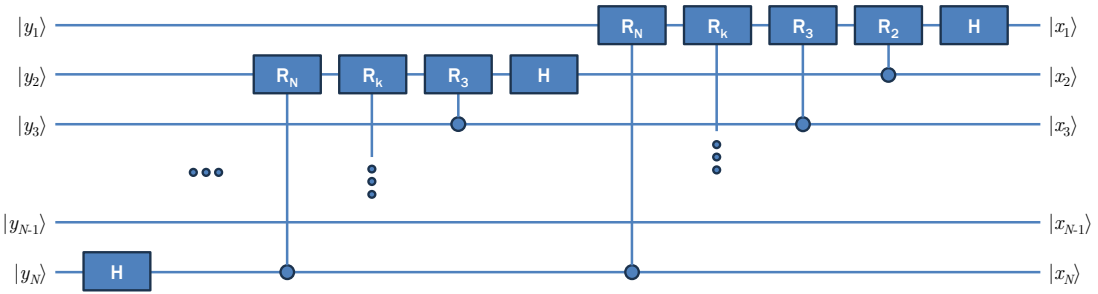


Figure 24: Quantum Inverse Fourier transform

Conclusion

Conclusion

- Non-Clifford set of quantum logic gates
- Measurement of quantum states
- Dynamic quantum circuits
- Limitation: Quantum algorithm consisting of only Clifford gates can be efficiently simulated by classical computers, yielding no performance boost
- Divergence: Quantum algorithms that include Non-Clifford gates cannot be efficiently simulated by classical computers, but still perform well on quantum computers

Questions?

Target Learning Outcomes/1

- Master Non-Clifford Quantum Operations
  1. Differentiate between Clifford and non-Clifford gate sets and explain the significance of the Gottesman-Knill Theorem regarding classical simulation efficiency.
  2. Demonstrate boolean universality by constructing classical logic gates (AND, OR, NOT, XOR, NAND) using the Toffoli (CCX) gate.
  3. Apply generalized rotations using RX, RY, and RZ gates to manipulate qubit amplitudes, signs, and phases.
  4. Synthesize universal unitaries by combining RZ, RX, and P gates to simulate any single-qubit operator.

Target Learning Outcomes/2

- Implement and Evaluate Measurement Strategies
  1. Distinguish between five types of state measurement: analytical solutions, projective measurement, state expectation, variance, and simulation (Sampling).
  2. Calculate probabilities and expectations for quantum states using Pauli projectors (X,Y,Z) and eigen-decomposition.
  3. Analyze outcome dispersion by computing the variance of quantum operators to determine the consistency of qubit distributions.
  4. Execute quantum simulations in PennyLane using both step-by-step sampling and end-to-end frequency counting.
- Model Quantum States via Density Matrices
  1. Convert pure and mixed states into density matrix representations.
  2. Verify state purity by applying the trace property $\mathrm{tr}(\rho^2) = 1$ for pure states versus $\mathrm{tr}(\rho^2) < 1$ for mixed states.
  3. Reconstruct qubits from the expectations of Pauli projectors using the density matrix formula.

Target Learning Outcomes/3

- Design Dynamic and Algorithmic Quantum Circuits
  1. Implement mid-circuit measurements (MCM) to extract classical information during an algorithm's execution.
  2. Construct conditional logic using `qml.cond` to trigger quantum operators based on measurement outcomes or external parameters.
  3. Apply postselection to refine algorithm results by discarding circuit runs that do not meet specific measurement criteria.
  4. Develop Iterative Algorithms using `for_loop` and `while_loop` decorators to automate complex structures like the GHZ state and Quantum Fourier Transform (QFT).