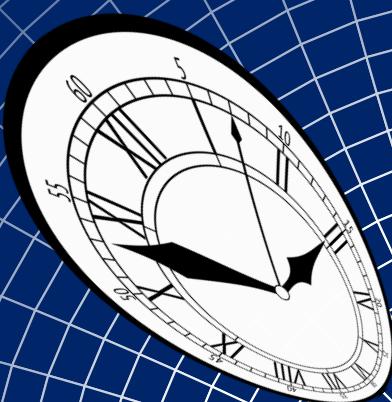


Qhronology

A Python package for resolving quantum time-travel paradoxes and performing general quantum information processing & computation

Documentation,
Examples,
and Theory



Lachlan G. Bishop

Qhronology

Release 1.0.0

Lachlan G. Bishop

June 2025

Qhronology: A Python package for resolving quantum time-travel paradoxes and performing general quantum information processing & computation.

© 2025 Lachlan G. Bishop

First published in June 2025 as version 1.0.0

Latest documentation PDF version available online:

https://github.com/lgbishop/qhronology/blob/latest/docs/_build/latex/Qhronology.pdf

Official website:

<https://qhronology.com>

Source code available online:

<https://github.com/lgbishop/qhronology>

This documentation, including its text and images, is published under the CC BY-NC-ND 4.0 license. In accordance with this license, you are free to share (copy and redistribute) the material in any medium or format under the following terms:

Attribution – You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial – You may not use the material for commercial purposes.

NoDerivatives – If you remix, transform, or build upon the material, you may not distribute the modified material.

No additional restrictions – You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

For more information, see:

<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Read the full license here:

<https://github.com/lgbishop/qhronology/blob/latest/licenses/CC-BY-NC-ND-4.0.txt>

Contents

1 Overview	1
1.1 Features	2
1.2 Package installation and structure	3
1.3 Examples	4
1.4 Documentation	8
1.5 License	8
1.6 Contributing	9
1.7 Citation	9
1.8 Possible future work	10
I Theory	11
2 Mathematical foundations of quantum mechanics	13
2.1 Vector spaces	13
2.2 Linear operators and operations	17
2.3 Composite systems	21
2.4 Superoperators	25
2.5 Concavity and convexity	25
3 Quantum mechanics on discrete Hilbert spaces	27
3.1 Quantum states	27
3.2 Quantum quantities	32
3.3 Separability and entanglement	35
3.4 Quantum operations	36
3.5 Quantum evolutions	37
3.6 Quantum measurements	37
3.7 Integrals over quantum states	39
4 Quantum mechanics on continuous Hilbert spaces	41
4.1 Probabilities and the Born rule	41
4.2 Time evolution	42
4.3 The path-integral formulation	43
5 Quantum circuitry	45
5.1 Anatomy of a quantum circuit	45
5.2 List of special gates	52
5.3 Examples	55
5.4 Remarks	56
6 Classical theories of time travel	59
6.1 Closed timelike curves and time machines	59
6.2 Time-travel paradoxes	60
6.3 The Cauchy problem on spacetimes containing CTCs	61
6.4 Time-travelling billiard balls	61
7 Quantum theories of time travel	63
7.1 The Deutsch model (D-CTCs)	64
7.2 Postselected teleportation (P-CTCs)	67
7.3 Comparison of D-CTCs and P-CTCs	71
7.4 Alternative formulations of quantum time travel	72

II Documentation	77
8 States	83
8.1 Main class	83
8.2 Subclasses	103
9 Gates	105
9.1 Main class	105
9.2 Subclasses	112
9.3 Combining gates	133
10 Circuits	139
10.1 Main class	139
11 Prescriptions	157
11.1 Main class	157
11.2 Subclasses	163
12 Matrices	175
12.1 Functions	175
13 Quantities	183
13.1 Functions	183
13.2 Mixin	187
14 Operations	189
14.1 Functions	189
14.2 Mixin	197
III Examples	199
15 Quantum algorithms and protocols	201
15.1 Generation of a Bell state	201
15.2 Generation of a GHZ state	203
15.3 Generation of a W state	205
15.4 CNOT (controlled-NOT)	207
15.5 CCNOT (controlled-controlled-NOT)	209
15.6 Toffoli decomposition	212
15.7 Half adder	215
15.8 Full adder	217
15.9 Ripple-carry adder	220
15.10 Ripple-carry adder (iterative)	225
15.11 Carry-lookahead adder	228
15.12 Fourier transform adder	234
15.13 Quantum teleportation	238
15.14 PSWAP (power-SWAP)	240
15.15 iSWAP (imaginary-SWAP)	242
15.16 Quantum state tomography	244
15.17 Quantum state tomography via weak measurements	245
16 Quantum closed timelike curves	251
16.1 Grandfather paradox	251
16.2 Unproven-theorem paradox	255
16.3 Billiard-ball paradox	258
16.4 Manual P-CTC	269
16.5 Equivalent-circuit picture for D-CTCs	271

IV Appendix	273
17 List of symbols	275
18 List of abbreviations	277
Bibliography	279
Index	297

Qhronology

A Python package for resolving quantum time-travel paradoxes and performing general quantum information processing & computation



Qhronology is a Python package for computing the states of the chronology-respecting (CR) and chronology-violating (CV) quantum systems according to the foremost quantum theories of time travel. It also functions as a (mostly) complete quantum circuit simulator, and contains an engine for the basic visualization of quantum circuit diagrams. Its main features include:

- ▶ temporal paradox resolution using quantum-mechanical prescriptions of time travel
 - Deutsch's model (D-CTCs)
 - postselected teleportation (P-CTCs)
- ▶ general quantum computation and information processing
 - algebraic, symbolic calculations
 - (classical) simulation of quantum experiments (limited by the finite precision of floating-point arithmetic)
- ▶ quantum circuit visualization
 - text-based semigraphical diagrams constructed using glyphs from monospace fonts (support for both ASCII and Unicode)

The primary purpose of Qhronology is to facilitate the study of quantum theories of time travel in both educational and research capacities. In addition to this, the package also functions as a quantum circuit simulator, allowing for comprehensive analysis of the general quantum operations of quantum information processing and computation. Qhronology's underlying mathematical system accomplishes this using the standard d -dimensional matrix mechanics of discrete-variable quantum theory in a general \mathbb{C}^d -representation.

The package provides a sufficiently complete and self-contained set of tools with the intention that performing transformations on quantum constructs and data with external packages and libraries need not be necessary (in most cases). Qhronology aims to make the expression of quantum states, gates, circuits, and time-travel prescriptions near-limitlessly possible with a framework that is syntactically simple, informationally dense, programmatically intuitive, mathematically powerful, extremely flexible, and easily extensible.

In addition to functionality provided by Python's standard library, Qhronology is built around features from both the canonical [SymPy¹](https://sympy.org) ([repository²](https://github.com/sympy/sympy)) and [NumPy³](https://numpy.org) ([repository⁴](https://github.com/numpy/numpy)) projects. In particular, the package greatly leverages the symbolic and linear algebra capabilities of the former, and so aims to have a deep compatibility with SymPy and its matrix objects. It is hoped that users possessing experience with these projects should therefore find Qhronology's interface both familiar and intuitive.

¹ <https://sympy.org>

² <https://github.com/sympy/sympy>

³ <https://numpy.org>

⁴ <https://github.com/numpy/numpy>

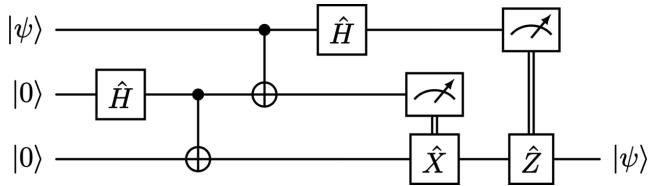
Note:

Qhronology in its current form is considered to be highly experimental. Its output may be incorrect, and some features may not work as intended. Additionally, please note that its functionality, including any and all functions, classes, methods, and modules, may be subject to change in future versions.

1.1 Features

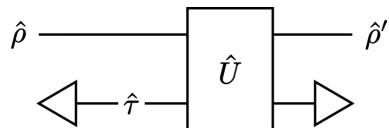
1.1.1 Quantum computing simulations

Designed to provide a powerful set of features with a simple and intuitive syntax, Qhronology facilitates the simulation of quantum computation, information processing, and algebraic calculations.



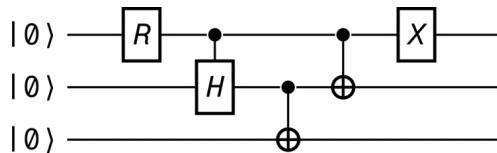
1.1.2 Quantum resolutions to antichronological time-travel paradoxes

The fundamental indeterminism of quantum mechanics can be leveraged to provide resolutions to quantum formulations of classic time-travel paradoxes (such as the infamous *grandfather paradox*). A select few prescriptions by which this may be achieved, including Deutsch's model (D-CTCs) and the postselected teleportation prescription (P-CTCs), are implemented both as bare functions and class methods.



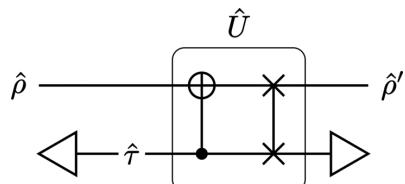
1.1.3 Quantum circuit visualization

Quantum circuit diagrams provide a powerful pictorialism through which any quantum process can be visualized as a network of quantum logic gates connected by wires. Qhronology provides this functionality for any quantum process constructed using its built-in classes.



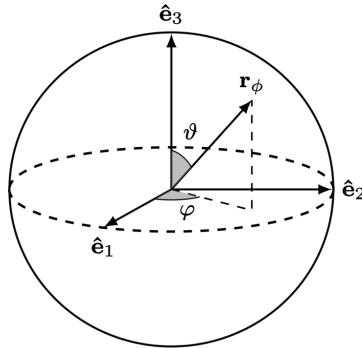
1.1.4 Numerous examples

Bundled with the project is a small collection of complete examples that showcase its capabilities and syntax. These are divided into two categories: *Quantum algorithms and protocols* (page 201) and *Quantum closed timelike curves* (page 251). The former contains implementations of canonical algorithms in quantum computing, while the latter consists of more exotic circuits that use quantum mechanics to resolve paradoxical scenarios of antichronological time travel.



1.1.5 Extensive documentation

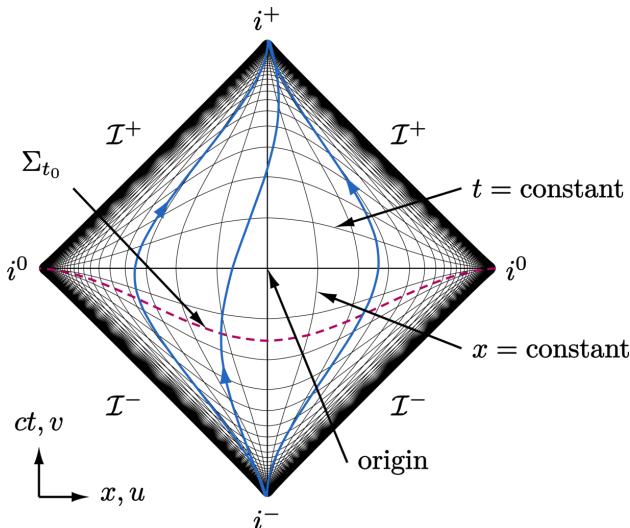
All of the functions, classes, and methods in each of the various submodules have been rigorously detailed in their respective sections within the documentation. This includes multiple examples of usage for each, aiding the user's understanding of every available feature.



1.1.6 Foundational theory

All of the underlying mathematics upon which Qhronology is built is presented as a series of pedagogical reference articles within the documentation. This includes sections on the mathematical foundations of quantum mechanics (Hilbert spaces, linear operators, composite systems, etc.), quantum theory on both discrete and continuous Hilbert spaces, a brief overview of the quantum circuitry picturalism, and physical theories of time travel (both classical and quantum).

The aim of this theory is to serve as a comprehensive and complete reference for basic quantum mechanics and physical theories of time travel, thereby enabling the keen user to embark upon further research into these fascinating areas of study.



1.2 Package installation and structure

Local installation of Qhronology from PyPI⁵ can be accomplished using pip (website⁶, repository⁷) via your operating system's command line, e.g.,

```
$ pip install qhronology
```

You may also be able to use an alternative package manager of your choice.

⁵ <https://pypi.org/project/qhronology/>

⁶ <https://pip.pypa.io/>

⁷ <https://github.com/pypa/pip>

After installation, the package can simply be imported in Python in the usual way. One suggestion is as follows:

```
import qhronology as qy
```

The package has the following directory structure:

```
qhronology
├── quantum
│   ├── circuits.py
│   ├── gates.py
│   ├── prescriptions.py
│   └── states.py
├── mechanics
│   ├── matrices.py
│   ├── operations.py
│   └── quantities.py
└── utilities (intended for internal use only)
    ├── classification.py
    ├── diagrams.py
    ├── helpers.py
    ├── objects.py
    └── symbolics.py
```

1.2.1 Requirements

Within the package and documentation, SymPy and NumPy are imported in their conventional manners:

```
import sympy as sp
import numpy as np
```

Qhronology is compatible with the following package versions (from [requirements.txt](#)⁸):

```
sympy>=1.12
numpy>=1.26
```

These are the earliest versions with which the current release has been tested, but older versions may also be compatible. It also requires

```
python>=3.11
```

1.3 Examples

1.3.1 Generation of a Bell state

Generation of the $|\Phi^+\rangle$ Bell state from primitive $|0\rangle$ states:

```
from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Hadamard, Not
from qhronology.quantum.circuits import QuantumCircuit

# Input
zero_state = VectorState(spec=[(1, [0])], label="0")

# Gates
```

(continues on next page)

⁸ <https://github.com/lgbishop/qhronology/blob/latest/requirements.txt>

(continued from previous page)

```

HI = Hadamard(targets=[0], num_systems=2)
CN = Not(targets=[1], controls=[0], num_systems=2)

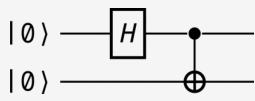
# Circuit
generator = QuantumCircuit(inputs=[zero_state, zero_state], gates=[HI, CN])
generator.diagram()

# Output
phi_plus = generator.state(label="Φ+")

# Results
phi_plus.print()

```

```
>>> generator.diagram()
```



```

>>> phi_plus.print()
|Φ+⟩ = sqrt(2)/2|0,0⟩ + sqrt(2)/2|1,1⟩

```

1.3.2 Quantum teleportation

Quantum teleportation of an arbitrary qubit $|\psi\rangle = a|0\rangle + b|1\rangle$:

```

from qchronology.quantum.states import VectorState
from qchronology.quantum.gates import Hadamard, Not, Measurement, Pauli
from qchronology.quantum.circuits import QuantumCircuit
from qchronology.mechanics.matrices import ket

# Input
teleporting_state = VectorState(
    spec=[["a", "b"]],
    symbols={"a": {"complex": True}, "b": {"complex": True}},
    conditions=[("a*conjugate(a) + b*conjugate(b)", "1")],
    label="ψ",
)
zero_state = VectorState(spec=[(1, [0, 0])], label="0,0")

# Gates
IHI = Hadamard(targets=[1], num_systems=3)
ICN = Not(targets=[2], controls=[1], num_systems=3)
CNI = Not(targets=[1], controls=[0], num_systems=3)
HII = Hadamard(targets=[0], num_systems=3)
IMI = Measurement(
    operators=[ket(0), ket(1)], observable=False, targets=[1], num_systems=3
)
MII = Measurement(
    operators=[ket(0), ket(1)], observable=False, targets=[0], num_systems=3
)
ICX = Pauli(index=1, targets=[2], controls=[1], num_systems=3)
CIZ = Pauli(index=3, targets=[2], controls=[0], num_systems=3)

```

(continues on next page)

(continued from previous page)

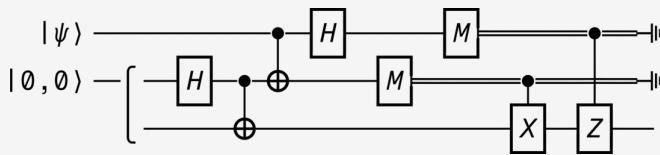
```
# Circuit
teleporter = QuantumCircuit(
    inputs=[teleporting_state, zero_state],
    gates=[IHI, ICN, CNI, HII, IMI, MII, ICX, CIZ],
    traces=[0, 1],
)
teleporter.diagram(force_separation=True)

# Output
teleported_state = teleporter.state(norm=1, label="ρ")

# Results
teleporting_state.print()
teleported_state.print()

print(teleporting_state.distance(teleported_state))
print(teleporting_state.fidelity(teleported_state))
```

```
>>> teleporter.diagram(force_separation=True)
```



```
>>> teleporting_state.print()
|ψ⟩ = a|0⟩ + b|1⟩
```

```
>>> teleported_state.print()
ρ = a*conjugate(a)|0⟩⟨0| + a*conjugate(b)|0⟩⟨1| + b*conjugate(a)|1⟩⟨0| +_
→b*conjugate(b)|1⟩⟨1|
```

```
>>> teleporting_state.distance(teleported_state)
0
```

```
>>> teleporting_state.fidelity(teleported_state)
1
```

1.3.3 Unproven-theorem paradox

Computing resolutions to the unproven-theorem paradox according to various prescriptions of quantum time travel (D-CTCs and P-CTCs):

```
from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Not, Swap
from qhronology.quantum.prescriptions import QuantumCTC, DCTC, PCTC

# Input
mathematician_state = VectorState(spec=[(1, [0])], label="0")
book_state = VectorState(spec=[(1, [0])], label="0")
```

(continues on next page)

(continued from previous page)

```

# Gates
NIC = Not(targets=[0], controls=[2], num_systems=3)
CNI = Not(targets=[1], controls=[0], num_systems=3)
IS = Swap(targets=[1, 2], num_systems=3)

# CTC
unproven = QuantumCTC(
    inputs=[mathematician_state, book_state],
    gates=[NIC, CNI, IS],
    systems_respecting=[0, 1],
)
unproven.diagram()

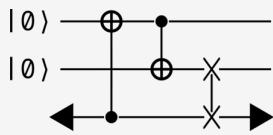
# Output
# D-CTCs
unproven_DCTC = DCTC(circuit=unproven)
unproven_DCTC_respecting = unproven_DCTC.state_respecting(norm=1, label=" $\rho_D$ ")
unproven_DCTC_violating = unproven_DCTC.state_violating(norm=1, label=" $\tau_D$ ")

# P-CTCs
unproven_PCTC = PCTC(circuit=unproven)
unproven_PCTC_respecting = unproven_PCTC.state_respecting(norm=1, label=" $\psi_P$ ")
unproven_PCTC_violating = unproven_PCTC.state_violating(norm=1, label=" $\tau_P$ ")

# Results
unproven_DCTC_respecting.print()
unproven_DCTC_violating.print()
unproven_PCTC_respecting.print()
unproven_PCTC_violating.print()

```

```
>>> unproven.diagram()
```



```
>>> unproven_DCTC_respecting.print()
 $\rho_D = g|0,0\rangle\langle 0,0| + (1 - g)|1,1\rangle\langle 1,1|$ 
```

```
>>> unproven_DCTC_violating.print()
 $\tau_D = g|0\rangle\langle 0| + (1 - g)|1\rangle\langle 1|$ 
```

```
>>> unproven_PCTC_respecting.print()
 $|\psi_P\rangle = \sqrt{2}/2|0,0\rangle + \sqrt{2}/2|1,1\rangle$ 
```

```
>>> unproven_PCTC_violating.print()
 $\tau_P = 1/2|0\rangle\langle 0| + 1/2|1\rangle\langle 1|$ 
```

1.4 Documentation

The latest version of the documentation for the package is available at:

- ▶ The official website: <https://qhrontology.com>
- ▶ The official PDF document: [Qhrontology.pdf](#)⁹

Both of these are built using [Sphinx](#)¹⁰ ([repository](#)¹¹), with their shared source files residing within the `docs` directory at the root of the project’s repository. This includes all project text and artwork. Please see `shell-sphinx.nix`¹² within that directory for a list of dependencies required to build both documentation targets. Note that a full LaTeX system installation from 2024 or later is required to build the project’s PDF documentation, figures, and artwork (including the logo). Also note that the documentation’s rendered circuit diagrams (generated from the package itself) were created using a custom LaTeX template (`render-text.tex`¹³) and associated shell script (`render-text.sh`¹⁴).

1.5 License

1.5.1 Package

The Qhrontology package (including any distributions, both source and built, and its source code) is dual-licensed, and you may only use it under the terms of a relevant license agreement. The particular one which applies to you depends on your use of the package and can be summarized as follows:

- ▶ **Non-commercial use:** You are free to copy, distribute, and transmit Qhrontology for non-commercial purposes. Non-commercial use is subject to the terms of version 3 of the GNU Affero General Public License (AGPL-3.0-or-later). Redistribution of the software in accordance with this license must only be done under the same non-commercial terms. The AGPL in its entirety can be viewed online at <https://www.gnu.org/licenses/agpl-3.0.html>. A copy of the license is also included with the project’s source code: [AGPL-3.0-or-later.txt](#)¹⁵.
- ▶ **Commercial use:** Any use of Qhrontology for a commercial purpose is subject to and requires a special license. If you intend to use the package in such a manner, please contact lachlanbishop@protonmail.com to arrange a license agreement.

1.5.1.1 Guidelines on distinguishing use

Whether a particular use of Qhrontology is considered to be either “non-commercial” or “commercial” depends on the use, not the user, and some guidelines are set out below:

- ▶ *Non-commercial* use is anticipated to primarily involve students and teachers who may wish to use Qhrontology at home and/or an accredited academic institution (e.g., school, university, etc.) for the purpose of education, without intending to seek any commercial advantage or financial gain. This includes teachers in schools and universities where tuition fees are charged, so long as the use of Qhrontology is limited to personal or individual classroom teaching or public academic research.
- ▶ *Commercial* use is anticipated to primarily involve publishers, online schools, online universities, and other organizations (both for-profit and non-profit) who wish to use Qhrontology to support activities that are intended toward securing a commercial advantage or the generation of revenue or monetary compensation. This includes, but is not limited to, any of the following:
 - The use of Qhrontology (and its source code) to generate or develop educational materials or resources which will be sold in exchange for a fee (including course or tuition fees) or (if given away for free) which are used to gain a commercial advantage for the user.

⁹ https://github.com/lgbishop/qhrontology/blob/latest/docs/_build/latex/Qhrontology.pdf

¹⁰ <https://www.sphinx-doc.org>

¹¹ <https://github.com/sphinx-doc/sphinx>

¹² <https://github.com/lgbishop/qhrontology/blob/latest/docs/shell-sphinx.nix>

¹³ <https://github.com/lgbishop/qhrontology/blob/latest/docs/source/figures/render-text.tex>

¹⁴ <https://github.com/lgbishop/qhrontology/blob/latest/licenses/AGPL-3.0-or-later.txt>

¹⁵ <https://github.com/lgbishop/qhrontology/blob/latest/licenses/AGPL-3.0-or-later.txt>

- The provision of training, support or editorial services that use or reference Qhronology in exchange for a fee.
- The use of Qhronology (and its source code) within a non-academic ebook, textbook, or journal, whether or not the publication is distributed for a fee. Please note that the use of Qhronology in publicly available academic papers and conferences is considered to be non-commercial use and so does not require a commercial license agreement.
- The use of Qhronology (and its source code) to assist in securing advertising or sponsorship revenues.
- The use of Qhronology (and its source code) for training machine learning models or artificial intelligence systems, including its incorporation into any dataset used for such purposes.

1.5.2 Documentation

The Qhronology project's accompanying documentation, including its text, images, scripts, code snippets, compiled outputs (e.g., HTML and PDF), and source files, are published under the CC-BY-NC-ND-4.0 license. For more information about your rights and restrictions under this license, please visit <https://creativecommons.org/licenses/by-nc-nd/4.0/>. A copy of the license is provided with the project's source code: [CC-BY-NC-ND-4.0.txt¹⁶](#).

1.5.3 Other

External assets licensed for distribution with the project consist of the following:

- ▶ Fonts - applicable fonts are redistributed (vendored and embedded) as per the terms of the SIL Open Font License: [OFL-1.1.txt¹⁷](#)
- ▶ MathJax - redistributed (vendored) as per the terms of the Apache License, Version 2.0: [Apache-2.0.txt¹⁸](#)
- ▶ three.js - redistributed (vendored) as per the terms of the MIT License: [MIT.txt¹⁹](#)

1.6 Contributing

Contributions to Qhronology (both the package and its documentation), including any features, fixes, and suggestions, are welcome provided they are compatible with the project's concept and vision, while also conforming to its style. Please see [CONTRIBUTING²⁰](#) for more details about contributing to the project. Feel free to contact lachlanbishop@protonmail.com to discuss any significant additions or changes you wish to propose.

1.7 Citation

```
@software{bishop_qhronology_2025,
  title = {{Qhronology: A Python package for resolving quantum time-travel paradoxes and performing general quantum information processing & computation}},
  author = {Bishop, Lachlan G.},
  year = {2025},
  url = {https://github.com/lgbishop/qhronology},
}
```

¹⁶ <https://github.com/lgbishop/qhronology/blob/latest/licenses/CC-BY-NC-ND-4.0.txt>

¹⁷ <https://github.com/lgbishop/qhronology/blob/latest/licenses/OFL-1.1.txt>

¹⁸ <https://github.com/lgbishop/qhronology/blob/latest/licenses/Apache-2.0.txt>

¹⁹ <https://github.com/lgbishop/qhronology/blob/latest/licenses/MIT.txt>

²⁰ <https://github.com/lgbishop/qhronology/blob/latest/CONTRIBUTING>

1.8 Possible future work

- ▶ Package:
 - Write proper (more formal) unit tests.
 - Permit more intuitive usage (i.e., summation and multiplication) of quantum objects via operator overloading.
 - Tighter integration with SymPy's `pprint()` functionality for enhanced state and gate printing.
 - Implement T-CTCs (the *transition-probabilities* quantum model of time travel).
 - Create the ability for circuit visualizations to target *Quantikz* LaTeX output.
 - ▷ Automatically rasterize using available (local) LaTeX installation.
 - Implement the permutation (PERM) gate.
- ▶ Documentation:
 - More examples.
 - Website:
 - ▷ Fix citation numbering.
- ▶ Theory:
 - Expand section on the Cauchy problem near CTCs.
 - Add a section on the general theory of relativity and the associated geometric theories of CTCs.

Part I

Theory

In this section, a brief but (mostly) complete introduction to quantum mechanics and its mathematical description is presented. This is intended to serve as a reference to the quantum mechanics implemented within the Qhronology package, in addition to providing a pedagogical summary of the basic physics relevant to this project. Included are discussions on antichronological time travel from the perspectives of both classical and quantum physics. The exposition on these topics is designed to be sufficiently comprehensive so as to provide a pathway for the determined reader to make a foray into the fascinating area of research that is the quantum mechanics of time travel.

Mathematical foundations of quantum mechanics

Quantum mechanics, one of the cornerstones of our modern view of reality, is founded on the mathematical formalism of linear algebra, which itself is built upon structures known as vector spaces. The following mathematical preliminaries have been adapted from [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12].

2.1 Vector spaces

A (linear) *vector space* over a field \mathbb{F} is a non-empty set \mathcal{V} which is closed under the operations of addition and scalar multiplication. This is to say that, for elements (termed *vectors*) $\psi, \phi, \chi \in \mathcal{V}$, closure under addition requires the following axioms to be satisfied:

- (i) (commutativity): $\psi + \phi = \phi + \psi,$
- (ii) (associativity): $\psi + (\phi + \chi) = (\psi + \phi) + \chi,$
- (iii) (identity): $\exists! \mathbf{0} \in \mathcal{V}$ such that $\psi + \mathbf{0} = \psi.$

Similarly, (2.1) given some scalars $a, b, c \in \mathbb{F}$, closure under multiplication with a scalar likewise requires

- (i) (distributivity over \mathcal{H}): $c(\psi + \phi) = c\psi + c\phi,$
- (ii) (distributivity in \mathbb{F}): $(a + b)\psi = a\psi + b\psi,$
- (iii) (identity): $\exists! 1 \in \mathbb{F}$ such that $1\psi = \psi.$

Formally, we say that \mathcal{V} is an \mathbb{F} -vector space if it is an (additive) Abelian group with a function (scalar multiplication) $\mathbb{F} \times \mathcal{V} \rightarrow \mathcal{V}$ which satisfies the above axioms.

A subset of vectors $\{e_i\}_{i=1}^d$ in \mathcal{V} is said to be *linearly independent* if and only if the equation

$$\sum_{i=1}^d c_i e_i = 0 \quad (2.3)$$

has the sole solution $c_i = 0$ for all i (where $c_i \in \mathbb{F}$). Otherwise, the set of vectors is said to be *linearly dependent*. The *dimension* of a vector space \mathcal{V} , denoted by $\dim(\mathcal{V})$, is defined as the least upper bound of linearly independent vectors in \mathcal{V} . Informally, this is the largest possible number of linearly independent vectors which reside in \mathcal{V} . A vector space is said to have infinite dimension if there is no such largest number.

Furthermore, a subset $\{e_i\}_{i=1}^d$ is said to *span* the vector space \mathcal{V} if and only if every vector $\psi \in \mathcal{V}$ can be expressed as a (finite) linear combination of the form

$$\psi = \sum_{i=1}^d \psi_i e_i \quad (2.4)$$

where $\psi_i \in \mathbb{F}$ collectively characterize the vector ψ . To this end, we often write

$$\psi = (\psi_i)_{i=1}^d = (\psi_1, \psi_2, \dots, \psi_d)^T. \quad (2.5)$$

Moreover, if $\{e_i\}_{i=1}^d$ is both linearly independent and spans \mathcal{V} , then this subset is a (vector) *basis* for \mathcal{V} , and its linear combinations are unique.

2.1.1 Inner product space

Let \mathcal{V} be a vector space over \mathbb{C} . Such a vector space is an *inner product space* if it is equipped with an *inner product*, which is defined as the positive-definite symmetric bilinear map

$$\langle \cdot, \cdot \rangle : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{C}. \quad (2.6)$$

that by definition obeys

- (i) (conjugate symmetry): $\langle \phi, \psi \rangle = \langle \psi, \phi \rangle^*$,
- (ii) (linearity): $\langle \phi, a\psi \rangle = a\langle \phi, \psi \rangle$,
- (iii) (additivity): $\langle \phi, \psi + \chi \rangle = \langle \phi, \psi \rangle + \langle \phi, \chi \rangle$,
- (iv) (positive-definiteness): $\langle \psi, \psi \rangle > 0$ if $\psi \neq \mathbf{0}$.

Here, z^* is the complex conjugate of $z \in \mathbb{C}$. Note that together (2.7)-(i) and (2.7)-(ii) imply that the definition is *antilinear* in its first argument, e.g., $\langle a\phi, \psi \rangle = a^*\langle \phi, \psi \rangle$, which aligns with the convention in the physics literature.

A pair of vectors $\psi, \phi \in \mathcal{V}$ are *orthogonal* if

$$\langle \phi, \psi \rangle = 0. \quad (2.8)$$

Furthermore, a set of vectors $\{\phi_i\}_{i=1}^d$ is said to be *orthonormal* with respect to the inner product (2.6) if

$$\langle \phi_i, \phi_j \rangle = \delta_{ij} \quad (2.9)$$

where

$$\delta_{ij} \equiv \begin{cases} 0 & \text{when } i \neq j; \\ 1 & \text{when } i = j. \end{cases} \quad (2.10)$$

is the *Kronecker delta*. Orthonormality is a useful property: given an orthonormal basis $\{e_i\}_{i=1}^d$, the coefficients in the linear expansion (2.4) may be (uniquely) determined by taking the inner product, i.e.,

$$\langle e_i, \psi \rangle = \left\langle e_i, \sum_{j=1}^d \psi_j e_j \right\rangle = \sum_{j=1}^d \psi_j \langle e_i, e_j \rangle = \psi_i. \quad (2.11)$$

Using this, we may rewrite the linear combination (2.4) as

$$\psi = \sum_{i=1}^d \langle e_i, \psi \rangle e_i, \quad (2.12)$$

with which we can express the inner product of two vectors as

$$\langle \phi, \psi \rangle = \sum_{i=1}^d \langle \phi, e_i \rangle \langle e_i, \psi \rangle = \sum_{i=1}^d \phi_i^* \psi_i. \quad (2.13)$$

2.1.2 Vector norm

The vector *p-norm* is a (non-negative-valued) map

$$\|\cdot\|_p : \mathcal{V} \rightarrow \mathbb{R} \quad (2.14)$$

which characterizes the notion of the “length” of a vector. With $p \geq 1$, it is defined for any $\psi \in \mathcal{V}$ as

$$\|\psi\|_p = \left[\sum_{i=1}^d |\psi_i|^p \right]^{\frac{1}{p}}, \quad (2.15)$$

and possesses the following properties:

- (i) (scalar multiplication): $\|c\psi\|_p = |c|\|\psi\|_p$,
- (ii) (positive-definiteness): $\|\psi\|_p > 0$ if $\psi \neq \mathbf{0}$,
- (iii) (triangle inequality): $\|\psi + \phi\|_p \leq \|\psi\|_p + \|\phi\|_p$.

Hölder's inequality provides a useful relation between the inner product of two vectors $\psi, \phi \in \mathcal{V}$ and their p -norms,

$$\langle \psi, \phi \rangle \leq \|\psi\|_p \|\phi\|_q, \quad \frac{1}{p} + \frac{1}{q} = 1. \quad (2.17)$$

A vector $\psi \in \mathcal{V}$ is said to be *normalized* if

$$\|\psi\|_p = 1 \quad (2.18)$$

for any $p \geq 0$.

An inner product is an important tool because it defines a way to multiply vectors together, with the result being a scalar. A useful consequence of this is our ability to define the 2-norm as

$$\|\psi\|_2 = \sqrt{\langle \psi, \psi \rangle}, \quad (2.19)$$

which, given an orthonormal basis $\{e_i\}_{i=1}^d$ and the expansion (2.13), yields *Parseval's identity*,

$$\|\psi\|_2 = \sqrt{\sum_{i=1}^d |\langle e_i, \psi \rangle|^2}. \quad (2.20)$$

Additionally, in the case if $p = 2$, Hölder's inequality (2.17) becomes the *Cauchy-Schwarz inequality*,

$$\langle \psi, \phi \rangle \leq \|\psi\|_2 \|\phi\|_2 = \sqrt{\langle \psi, \psi \rangle} \sqrt{\langle \phi, \phi \rangle}. \quad (2.21)$$

2.1.3 Dual space

Any vector space \mathcal{V} has a corresponding *dual vector space* \mathcal{V}^* , which is the space of all linear forms (functionals) on \mathcal{V} . Mathematically, this is to say that an element $\varphi \in \mathcal{V}^*$ defines a map

$$\varphi : \psi \mapsto \varphi(\psi) \in \mathbb{C} \quad (2.22)$$

for every $\psi \in \mathcal{V}$ such that

$$\varphi(a\psi + b\chi) = a\varphi(\psi) + b\varphi(\chi) \quad (2.23)$$

for all ψ, χ and $a, b \in \mathbb{C}$. Elements of the dual space \mathcal{V}^* are often termed *covectors*, and the space itself becomes a vector space when equipped with addition and scalar multiplication, that is,

- (i) (additivity): $(\varphi_1 + \varphi_2)(\psi) = \varphi_1(\psi) + \varphi_2(\psi)$,
- (ii) (scalar multiplication): $(c\varphi)(\psi) = c\varphi(\psi)$,

for all $\varphi, \varphi_1, \varphi_2 \in \mathcal{V}^*$, $\psi \in \mathcal{V}$, and $c \in \mathbb{C}$.

The most straightforward way of constructing linear maps in a dual space is to use the inner product: given some vector $\phi \in \mathcal{V}$, we may define a map $\langle \phi, \cdot \rangle \in \mathcal{V}^*$ which has the action

$$\langle \phi, \cdot \rangle : \psi \mapsto \langle \phi, \psi \rangle. \quad (2.25)$$

Being based on the inner product, this form is guaranteed to be linear in its argument.

2.1.4 Hilbert space

An inner product vector space equipped with a norm is called a *normed space*. A sequence $(\psi_i)_{n=1}^{\infty}$ in such a space is termed a *Cauchy sequence* if

$$\|\psi_i - \psi_j\|_p \rightarrow 0 \quad \text{as } i, j \rightarrow \infty. \quad (2.26)$$

If all Cauchy sequences converge (as per the above condition) in \mathcal{V} , then the normed space \mathcal{V} is said to be *complete*. This is to say that there exists a vector $\psi \in \mathcal{V}$ such that

$$\lim_{i \rightarrow \infty} \|\psi_i - \psi\|_p = 0. \quad (2.27)$$

A complete normed space is called a *Banach space*.

The special case of a normed inner product space that is complete with respect to the 2-norm (2.19) is known as a *Hilbert space*. This construction is central to the formalism of quantum theory. The *Riesz representation theorem* establishes that any linear map $\phi : \mathcal{H} \rightarrow \mathbb{C}$ can be *uniquely* written as (2.25) for some fixed choice of $\phi \in \mathcal{H}$. Consequently, the inner product provides a vector space isomorphism

$$\mathcal{H}^* \cong \mathcal{H}, \quad (2.28)$$

regardless of the dimensionality of \mathcal{H} . Importantly, this holds true for infinite-dimensional Hilbert spaces, which makes them especially easy to work with.

The simplest Hilbert space \mathcal{H} is one which has a finite number of dimensions $d \in \mathbb{Z}^+$. In this case, we have $\mathcal{H} \cong \mathbb{C}^d$, and such a space is always isomorphic to one with the inner product (2.13) and corresponding norm (2.20). In physics, finite-dimensional Hilbert spaces often arise as idealized descriptions of phenomena that would otherwise require more complex treatment. This usually appears where one is concerned with just a finite-dimensional subspace of a larger physical system.

Infinite-dimensional Hilbert spaces on the other hand are slightly more nuanced. A simple generalization to such dimensionality involves the space of infinite sequences of complex numbers $\psi \equiv (\psi_i)_{i=1}^{\infty}$ with converging norm, i.e.,

$$\|\psi\|_2 = \sqrt{\sum_{i=1}^{\infty} |\psi_i|^2} < \infty. \quad (2.29)$$

If this holds, then $\psi \in \ell^2$ where ℓ^2 is the space of *square-summable sequences*, which heuristically can be thought of as $\mathcal{H} = \ell^2 \cong \mathbb{C}^{\infty}$. The associated inner product is then simply the generalization of (2.13),

$$\langle \phi, \psi \rangle = \sum_{i=1}^{\infty} \phi_i^* \psi_i, \quad (2.30)$$

which, as per the Cauchy-Schwarz inequality (2.21), converges provided the individual vector norms do. Importantly, the notion of completeness of ℓ^2 with respect to its norm enables us to perform calculus in this space, including computing limits and derivatives of vectors $\psi \in \ell^2$. Generalizations to other sequence spaces, e.g., ℓ^p with $p \geq 1$ and the associated norm

$$\|\psi\|_p = \left[\sum_{i=1}^{\infty} |\psi_i|^p \right]^{\frac{1}{p}} < \infty. \quad (2.31)$$

are of course possible, but note that only in the case of $p = 2$ is the sequence space a Hilbert space.

2.1.5 Dirac notation

Here we establish the notational convention that is used nearly universally in the formalism of quantum mechanics on Hilbert spaces. This standard system is known as *Dirac notation* or *bra-ket notation*. In this notation, a vector $\psi \in \mathcal{H}$ is represented by $|\psi\rangle$ and is called a *ket*. Similarly, a covector in the dual space $\phi \in \mathcal{H}^*$ is denoted by $\langle \phi |$ and called a *bra*. With these constructs, the value of the dual map (2.25) is given by

$$\phi(\psi) \equiv \langle \phi | \psi \rangle \equiv \langle \phi, \psi \rangle. \quad (2.32)$$

This form is known as a *bra-ket*, and is simply a representation of the inner product in Dirac notation. Note that this implicitly uses the isomorphism (2.28) provided by the inner product, meaning that we have the antilinear correspondence between bras and kets,

$$\lambda^* \langle \psi | \longleftrightarrow \lambda |\psi \rangle. \quad (2.33)$$

Given an orthonormal basis $\{|e_i\rangle\}$ of a discrete Hilbert space \mathcal{H} , we can expand in Dirac notation a general ket $|\psi\rangle \in \mathcal{H}$ in terms of the basis as

$$|\psi\rangle = \sum_i \psi_i |e_i\rangle, \quad \psi_i = \langle e_i | \psi \rangle. \quad (2.34)$$

This notation is slightly cumbersome, and so it is conventional to simplify it by making the basis identification

$$|e_i\rangle \equiv |i\rangle, \quad (2.35)$$

with which the above expansion appears as

$$|\psi\rangle = \sum_i \langle i | \psi \rangle |i\rangle. \quad (2.36)$$

The alternative notation for the basis $\{|i\rangle\}$ is known as the *standard basis* (in the nomenclature of mathematics) or the *computational basis* (in the nomenclature of quantum computing). It is always assumed to be orthonormal, and necessarily spans \mathcal{H} . As these objects (kets) reside in a d -dimensional Hilbert space $\mathcal{H} \cong \mathbb{C}^d$, they have corresponding representations in \mathbb{C}^d as column vectors, the standard choice being explicitly

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \dots \quad |d-1\rangle = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}. \quad (2.37)$$

Here, the i -th element (indexing from zero, as customary in computer science) of the i -th vector contains a 1, while all of its other entries are 0. The corresponding covectors (bras) in the dual space $\mathcal{H}^* \cong \mathbb{C}^{*d}$ have similar representations as row vectors,

$$\begin{aligned} \langle 0 | &= [1 \ 0 \ \dots \ 0], \\ \langle 1 | &= [0 \ 1 \ \dots \ 0], \\ &\vdots \\ \langle d-1 | &= [0 \ 0 \ \dots \ 1]. \end{aligned} \quad (2.38)$$

In a finite-dimensional Hilbert space, any basis of linearly independent vectors has the same (finite) number of members, irrespective of their orthogonality.

2.2 Linear operators and operations

A *linear operator* \hat{A} on a Hilbert space \mathcal{H} is a map

$$\hat{A} : \text{dom}(\hat{A}) \rightarrow \text{ran}(\hat{A}) \quad (2.39)$$

which preserves linear combinations of vectors $|\psi\rangle, |\phi\rangle \in \mathcal{H}$, that is,

$$\hat{A}(\mu|\psi\rangle + \nu|\phi\rangle) = \mu\hat{A}|\psi\rangle + \nu\hat{A}|\phi\rangle, \quad (2.40)$$

for all $\mu, \nu \in \mathbb{C}$. Here, the space $\text{dom}(\hat{A}) \subseteq \mathcal{H}$ denotes the domain of \hat{A} while $\text{ran}(\hat{A}) \subseteq \mathcal{H}$ denotes its range. For an infinite-dimensional \mathcal{H} , operators are often not defined on the whole space, and so $\text{dom}(\hat{A})$ may be a proper subset of \mathcal{H} . For finite-dimensional Hilbert spaces on the other hand, linear operators may always be defined on the entire space such that $\text{dom}(\hat{A}) = \text{ran}(\hat{A}) = \mathcal{H}$. The space of all linear operators on \mathcal{H} is denoted by $\mathcal{L}(\mathcal{H})$. All the operators we will discuss are linear, and so we will henceforth omit the “linear” descriptor.

2.2.1 Bounded operators

An operator \hat{A} is said to be *bounded* if $\text{dom}(\hat{A}) = \mathcal{H}$ and there exists some fixed $M > 0$ such that for all $|\psi\rangle \in \mathcal{H}$,

$$\|\hat{A}|\psi\rangle\|_2 \leq M\||\psi\rangle\|_2. \quad (2.41)$$

Otherwise, the operator is said to be *unbounded*. Bounded (linear) operators preserve the normalizability of vectors, that is, they map normalizable vectors to vectors that are likewise normalizable. This means that they necessarily act on the whole Hilbert space \mathcal{H} , and the space of such bounded operators is denoted $\mathcal{B}(\mathcal{H})$.

2.2.2 Operator algebra

Given two such operators \hat{A} and \hat{B} , we can define their sum to be

$$(\alpha\hat{A} + \beta\hat{B}) : |\psi\rangle \mapsto \alpha\hat{A}|\psi\rangle + \beta\hat{B}|\psi\rangle \quad (2.42)$$

for all $\alpha, \beta \in \mathbb{C}$ and $|\psi\rangle \in \mathcal{H}$. We can also define their product to be the composition

$$\hat{A}\hat{B} : |\psi\rangle \mapsto \hat{A} \circ \hat{B}|\psi\rangle = \hat{A}(\hat{B}|\psi\rangle) \quad (2.43)$$

for all $|\psi\rangle \in \mathcal{H}$. It is easy to prove that both of these definitions provide forms that are linear in the original sense (2.39), and thus operators form an algebra.

In general, operators do not commute, e.g., $\hat{A}\hat{B} \neq \hat{B}\hat{A}$, and so the associated algebra is non-Abelian, or non-commutative. The difference between these orderings is captured by the *commutator*,

$$[\hat{A}, \hat{B}] = \hat{A}\hat{B} - \hat{B}\hat{A}, \quad (2.44)$$

which possesses the following properties:

- (i) (antisymmetry): $[\hat{A}, \hat{B}] = -[\hat{B}, \hat{A}]$,
- (ii) (linearity): $[\alpha_1\hat{A}_1 + \alpha_2\hat{A}_2, \hat{B}] = \alpha_1[\hat{A}_1, \hat{B}] + \alpha_2[\hat{A}_2, \hat{B}] \quad \forall \alpha_1, \alpha_2 \in \mathbb{C}$,
- (iii) (Leibniz identity): $[\hat{A}, \hat{B}\hat{C}] = [\hat{A}, \hat{B}]\hat{C} + \hat{B}[\hat{A}, \hat{C}]$,
- (iv) (Jacobi identity): $[\hat{A}, [\hat{B}, \hat{C}]] + [\hat{B}, [\hat{C}, \hat{A}]] + [\hat{C}, [\hat{A}, \hat{B}]] = 0$.

2.2.3 Outer product

In addition to the inner product, we may also use the Dirac notation to easily express the *outer product* of a vector $|\psi\rangle \in \mathcal{H}_1$ and a covector $\langle\phi| \in \mathcal{H}_2^*$,

$$\begin{aligned} |\psi\rangle\langle\phi| &\equiv |\psi\rangle \otimes \langle\phi| \\ &= \sum_{i,j} \langle i|\psi\rangle \langle\phi|j\rangle |i\rangle\langle j| \\ &= \sum_{i,j} \psi_i \phi_j^* |i\rangle\langle j|. \end{aligned} \quad (2.46)$$

Here, $\phi \otimes \psi$ denotes the tensor product between two vectors ϕ and ψ (see *Composite systems* (page 21)). The resulting form $|\psi\rangle\langle\phi| \in \mathcal{H}$ is a linear operator in its own right and exists in the space

$$\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2^*, \quad (2.47)$$

which itself is a Hilbert space.

2.2.4 Eigenvectors and eigenvalues

A vector $|\psi\rangle \in \mathcal{H}$ is said to be an *eigenvector* of an operator \hat{A} if

$$\hat{A}|\psi\rangle = a_\psi|\psi\rangle \quad (2.48)$$

where the number $a_\psi \in \mathbb{C}$ is called the *eigenvalue*. Given this definition, we can single out two important operators:

- The *identity* operator \hat{I} is the (unique) operator that satisfies

$$\hat{I}|\psi\rangle = |\psi\rangle, \quad \forall |\psi\rangle \in \mathcal{H}. \quad (2.49)$$

Given any orthonormal basis $\{|i\rangle\}$, we may write

$$\hat{I} = \sum_i |i\rangle\langle i|. \quad (2.50)$$

- The *zero* operator $\hat{0}$ is the (unique) operator that satisfies

$$\hat{0}|\psi\rangle = \mathbf{0}, \quad \forall |\psi\rangle \in \mathcal{H}, \quad (2.51)$$

where $\mathbf{0}$ is the zero vector (i.e., the additive identity).

An operator \hat{A} is called *invertible* if an operator \hat{B} exists such that

$$\hat{A}\hat{B} = \hat{B}\hat{A} = \hat{I}. \quad (2.52)$$

If such a \hat{B} exists, then it is the (unique) inverse of \hat{A} , and we denote it by $\hat{B} = \hat{A}^{-1}$. The *spectrum* of \hat{A} is the set of all complex numbers λ for which the operator $\hat{A} - \lambda\hat{I}$ fails to be invertible. In a finite-dimensional Hilbert space, the spectrum of \hat{A} is simply the set of eigenvalues of \hat{A} . The same is not true in general for infinite dimensions. The number of linearly independent eigenvectors possessing the same eigenvalue is called the *degeneracy* of that eigenvalue. Additionally, the *support* of an operator \hat{A} is the vector space spanned by the eigenvectors of \hat{A} with non-zero eigenvalues.

2.2.5 Transposition and conjugation

Given a finite-dimensional Hilbert space \mathcal{H} , each operator $\hat{A} \in \mathcal{L}(\mathcal{H})$ can be expanded in an orthonormal basis as

$$\hat{A} = \sum_{i,j} |i\rangle\langle i| \hat{A} |j\rangle\langle j| = \sum_{i,j} A_{ij} |i\rangle\langle j| \quad (2.53)$$

where

$$A_{ij} \equiv \langle i|\hat{A}|j\rangle \quad (2.54)$$

are the elements of the corresponding matrix representation of the operator (in the given basis). Similarly, the product of two operators may be written as

$$\hat{A}\hat{B} = \sum_{i,j,k} A_{ik} B_{kj} |i\rangle\langle j| \quad (2.55)$$

where $B_{kj} \equiv \langle k|\hat{B}|j\rangle$. These expressions make it easy to characterize the action of operations on such operators. Our primary interest lies in the *transpose* and the *conjugate transpose* (or *adjoint*), which are defined for any orthonormally expandable \hat{A} by

$$\begin{aligned} \text{transpose: } \hat{A}^T &= \sum_{i,j} A_{ij} |j\rangle\langle i|, \\ \text{conjugate transpose: } \hat{A}^\dagger &= \sum_{i,j} A_{ij}^* |j\rangle\langle i| = (\hat{A}^T)^* = (\hat{A}^*)^T. \end{aligned} \quad (2.56)$$

From these, it is easy to deduce the following properties:

$$\begin{aligned}
 \text{(i): } & (\hat{A}^T)^T = \hat{A} & (\hat{A}^\dagger)^\dagger = \hat{A}, \\
 \text{(ii): } & (\hat{A}\hat{B})^T = \hat{B}^T\hat{A}^T & (\hat{A}\hat{B})^\dagger = \hat{B}^\dagger\hat{A}^\dagger, \\
 \text{(iii): } & (\alpha\hat{A})^T = \alpha\hat{A}^T & (\alpha\hat{A})^\dagger = \alpha^*\hat{A}^\dagger, \\
 \text{(iv): } & (\hat{A} + \hat{B})^T = \hat{A}^T + \hat{B}^T & (\hat{A} + \hat{B})^\dagger = \hat{A}^\dagger + \hat{B}^\dagger.
 \end{aligned} \tag{2.57}$$

Note also that the conjugate transpose of the eigenvalue equation $\hat{A}|\psi\rangle = a_\psi|\psi\rangle$ is

$$\langle\psi|\hat{A}^\dagger = \langle\psi|a_\psi^* \tag{2.58}$$

where \hat{A}^\dagger is taken to act to the left on the covector $\langle\psi|$.

2.2.6 Trace

Another useful operation is the *trace*,

$$\text{tr}[\cdot] : \mathcal{L}(\mathcal{H}) \rightarrow \mathbb{C}, \tag{2.59}$$

defined on an operator $\hat{A} \in \mathcal{L}(\mathcal{H})$ as

$$\text{tr}[\hat{A}] \equiv \sum_i \langle i | \hat{A} | i \rangle = \sum_i A_{ii} \tag{2.60}$$

where $\{|i\rangle\}$ is any orthonormal basis. The trace has the following properties:

$$\begin{aligned}
 \text{(i): } & \text{tr}[\hat{A} + \hat{B}] = \text{tr}[\hat{A}] + \text{tr}[\hat{B}], \\
 \text{(ii): } & \text{tr}[\hat{A}\hat{B}] = \text{tr}[\hat{B}\hat{A}], \\
 \text{(iii): } & \text{tr}[\hat{A}^\dagger] = \text{tr}[\hat{A}]^*, \\
 \text{(iv): } & \text{tr}[|\psi\rangle\langle\psi|\hat{A}] = \langle\psi|\hat{A}|\psi\rangle.
 \end{aligned} \tag{2.61}$$

In a matrix representation of linear operators, the trace is simply the sum of the elements on the main diagonal of an operator's corresponding matrix form.

2.2.7 Normal operators

An important class of operators are the *normal* operators, which are characterized by their commutation with their own conjugate transpose, i.e.,

$$\hat{N}^\dagger \hat{N} = \hat{N} \hat{N}^\dagger. \tag{2.62}$$

The *spectral theorem* states that any normal operator \hat{N} admits a suitable *eigendecomposition*, that is, the eigenvectors $\{|v_i\rangle\}$ of \hat{N} form an orthonormal basis, and can be used in conjunction with the corresponding eigenvalues $\{\lambda_i\}$ to allow us to write

$$\hat{N} = \sum_i \lambda_i |v_i\rangle\langle v_i|, \quad \lambda_i \in \mathbb{C}. \tag{2.63}$$

The converse holds true, such that any orthogonally diagonalizable operator is normal. A useful consequence of this theorem is that it is easy to compute the function of an operator,

$$f(\hat{N}) = \sum_i f(\lambda_i) |v_i\rangle\langle v_i|. \tag{2.64}$$

Normal operators arise in many special forms, including:

- *Hermitian* (or *self-adjoint*) operators \hat{H} , defined by

$$\hat{H}^\dagger = \hat{H}. \tag{2.65}$$

If \hat{H} is Hermitian, then

$$\langle\psi|\hat{H}|\psi\rangle \in \mathbb{R}, \quad \forall |\psi\rangle \in \mathcal{H}, \tag{2.66}$$

that is, \hat{H} has only real eigenvalues.

- *skew-Hermitian* operators \hat{S} , defined by

$$\hat{S}^\dagger = -\hat{S}. \quad (2.67)$$

- *Unitary* operators \hat{U} , defined by

$$\hat{U}^\dagger \hat{U} = \hat{U} \hat{U}^\dagger = \hat{I}, \quad (2.68)$$

or equivalently,

$$\hat{U}^\dagger = \hat{U}^{-1}. \quad (2.69)$$

The characterizing nature of unitary operators is their preservation of the inner product, i.e.,

$$\langle \phi | \hat{U}^\dagger \hat{U} | \psi \rangle = \langle \phi | \psi \rangle, \quad \forall |\psi\rangle, |\phi\rangle \in \mathcal{H}, \quad (2.70)$$

which motivates the given definition.

- *Positive* operators \hat{P} , defined by

$$\langle \psi | \hat{P} | \psi \rangle \geq 0, \quad \forall |\psi\rangle \in \mathcal{H}. \quad (2.71)$$

Operators that fulfill this condition are said to be *positive-semidefinite*, and they may be constructed from any $\hat{A} \in \mathcal{L}(\mathcal{H})$ as

$$\hat{P} = \hat{A}^\dagger \hat{A}, \quad (2.72)$$

or using the spectral theorem,

$$\hat{P} = \sum_i P_i |v_i\rangle \langle v_i| \quad (2.73)$$

given eigenvalues $P_i \geq 0$ and eigenvectors $\{|v_i\rangle\}$. Note that \hat{P} is called *positive-definite* if

$$\langle \psi | \hat{P} | \psi \rangle > 0, \quad \forall |\psi\rangle \in \mathcal{H}. \quad (2.74)$$

In the special case where

$$\text{tr}[\hat{P}] = \sum_i P_i = 1, \quad (2.75)$$

then \hat{P} is called a *density* operator.

- *Projection* operators $\hat{\Pi}$, defined by

$$\hat{\Pi}^2 = \hat{\Pi} = \hat{\Pi}^\dagger, \quad (2.76)$$

which is to say that $\hat{\Pi}$ is both idempotent and Hermitian.

- *Involutory* operators \hat{Q} , defined by

$$\hat{Q}^2 = \hat{I}. \quad (2.77)$$

Involutory operators include those which are both Hermitian and unitary.

2.3 Composite systems

Hilbert spaces are vector spaces over \mathbb{C} , so we can construct composite spaces and the corresponding tensor product in much the same way as we would for less equipped spaces. We will first discuss the *bipartite* case, which may be easily generalized to larger composite systems (termed *multipartite*). Let \mathcal{H}_1 and \mathcal{H}_2 be a pair of finite-dimensional Hilbert spaces, with bases $\{|\mu_m\rangle\}_{m=1}^M$ and $\{|\nu_n\rangle\}_{n=1}^N$, respectively.

The tensor product space $\mathcal{H}_1 \otimes \mathcal{H}_2$ is spanned by all pairs of elements $|\mu_m\rangle \otimes |\nu_n\rangle$ chosen from the two bases. Note in particular that $\{|\mu_m\rangle \otimes |\nu_n\rangle\}_{m,n}$ is an orthonormal basis of the composite space. Thus, if we have the linear combinations,

$$\begin{aligned} |\psi\rangle &= \sum_{m=1}^M \psi_m |\mu_m\rangle, \\ |\phi\rangle &= \sum_{n=1}^N \phi_n |\nu_n\rangle, \end{aligned} \quad (2.78)$$

for any $|\psi\rangle \in \mathcal{H}_1$ and $|\phi\rangle \in \mathcal{H}_2$, we can formally express their *tensor product* as

$$|\psi\rangle \otimes |\phi\rangle = \sum_{m,n} \psi_m \phi_n |\mu_m\rangle \otimes |\nu_n\rangle. \quad (2.79)$$

This is often equivalently written as $|\psi\rangle|\phi\rangle$, $|\psi, \phi\rangle$, or even simply $|\psi\phi\rangle$. It follows that

$$\dim(\mathcal{H}_1 \otimes \mathcal{H}_2) = \dim(\mathcal{H}_1) \times \dim(\mathcal{H}_2), \quad (2.80)$$

and for any elements $|\psi\rangle, |\phi\rangle \in \mathcal{H}_1$ and $|\psi'\rangle, |\phi'\rangle \in \mathcal{H}_2$ we have the rules:

- (i) (linearity in \mathcal{H}_1): $(|\psi\rangle + |\phi\rangle) \otimes |\psi'\rangle = |\psi\rangle \otimes |\psi'\rangle + |\phi\rangle \otimes |\psi'\rangle$,
- (ii) (linearity in \mathcal{H}_2): $|\psi\rangle \otimes (|\psi'\rangle + |\phi'\rangle) = |\psi\rangle \otimes |\psi'\rangle + |\psi\rangle \otimes |\phi'\rangle$,
- (iii) (scalar multiplication): $\lambda(|\psi\rangle \otimes |\psi'\rangle) = (\lambda|\psi\rangle) \otimes |\psi'\rangle = |\psi\rangle \otimes (\lambda|\psi'\rangle)$.

Thus, the composite space $\mathcal{H}_1 \otimes \mathcal{H}_2$ is a vector space over \mathbb{C} , and becomes a Hilbert space when we endow it with an inner product. This is accomplished in the obvious way, where if $|\Psi\rangle, |\Phi\rangle \in \mathcal{H}_1 \otimes \mathcal{H}_2$ are two composite forms defined in general as

$$\begin{aligned} |\Psi\rangle &= \sum_{m,n} \Psi_{m,n} |\mu_m\rangle \otimes |\nu_n\rangle, \\ |\Phi\rangle &= \sum_{m,n} \Phi_{m,n} |\mu_m\rangle \otimes |\nu_n\rangle, \end{aligned} \quad (2.82)$$

then their inner product is

$$\begin{aligned} \langle \Psi | \Phi \rangle &= \sum_{i,j,k,l} \Phi_{ij}^* \Psi_{kl} (\langle \mu_i | \otimes \langle \nu_j |) (\langle \mu_k | \otimes \langle \nu_l |) \\ &= \sum_{i,j,k,l} \Phi_{ij}^* \Psi_{kl} \langle \mu_i | \mu_k \rangle_1 \langle \nu_j | \nu_l \rangle_2 \\ &= \sum_{i,j} \Phi_{ij}^* \Psi_{ij} \end{aligned} \quad (2.83)$$

by the orthonormality of the bases. Here, $\langle \cdot | \cdot \rangle_1$ and $\langle \cdot | \cdot \rangle_2$ denote the inner products on \mathcal{H}_1 and \mathcal{H}_2 , respectively. In the special case where the composite vectors are simple tensor products, e.g.,

$$\begin{aligned} |\Psi\rangle &= |\psi\rangle \otimes |\psi'\rangle, \\ |\Phi\rangle &= |\phi\rangle \otimes |\phi'\rangle, \end{aligned} \quad (2.84)$$

then their inner product becomes

$$\langle \Phi | \Psi \rangle = (\langle \phi | \otimes \langle \phi' |) (\langle \psi | \otimes \langle \psi' |) = \langle \phi | \psi \rangle_1 \langle \phi' | \psi' \rangle_2. \quad (2.85)$$

Systems which are to be understood as not being constructable from tensor products of Hilbert spaces (subsystems) are said to be *non-composite* and are often termed *unipartite*.

As far as linear operators on composite Hilbert spaces are concerned, if $\hat{A} \in \mathcal{L}(\mathcal{H}_A)$ and $\hat{B} \in \mathcal{L}(\mathcal{H}_B)$ are two operators, then their tensor product

$$\hat{A} \otimes \hat{B} \in \mathcal{L}(\mathcal{H}_A) \otimes \mathcal{L}(\mathcal{H}_B) = \mathcal{L}(\mathcal{H}_A \otimes \mathcal{H}_B) \quad (2.86)$$

is defined by the action on the product state $|\psi\rangle \otimes |\phi\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$ as

$$(\hat{A} \otimes \hat{B})(|\psi\rangle \otimes |\phi\rangle) = \hat{A}|\psi\rangle \otimes \hat{B}|\phi\rangle. \quad (2.87)$$

This has the following properties:

- (i): $(\hat{A}_1 + \hat{A}_2) \otimes \hat{B} = \hat{A}_1 \otimes \hat{B} + \hat{A}_2 \otimes \hat{B}$,
- (ii): $\hat{A} \otimes (\hat{B}_1 + \hat{B}_2) = \hat{A} \otimes \hat{B}_1 + \hat{A} \otimes \hat{B}_2$,
- (iii): $(\hat{A} \otimes \hat{B})^\dagger = \hat{A}^\dagger \otimes \hat{B}^\dagger$,
- (iv): $\text{tr}[\hat{A} \otimes \hat{B}] = \text{tr}[\hat{A}]\text{tr}[\hat{B}]$,
- (v): $(\lambda\hat{A}) \otimes \hat{B} = \hat{A} \otimes (\lambda\hat{B})$.

In general, an operator $\hat{Q} \in \mathcal{L}(\mathcal{H}_A \otimes \mathcal{H}_B)$ may be expanded as

$$\hat{Q} = \sum_{i,j,k,l} Q_{ik}^{jl} |i\rangle\langle j| \otimes |k\rangle\langle l|, \quad (2.89)$$

where $\{|i\rangle\}, \{|k\rangle\}$ are orthonormal bases in \mathcal{H}_A and $\{|j\rangle\}, \{|l\rangle\}$ are orthonormal bases in \mathcal{H}_B . The partial transposes of such an operator over each space \mathcal{H}_A and \mathcal{H}_B are defined respectively as

$$\begin{aligned} \hat{Q}^{\text{T}_A} &= \sum_{i,j,k,l} Q_{ik}^{jl} (|i\rangle\langle j|)^T \otimes |k\rangle\langle l| = \sum_{i,j,k,l} Q_{ik}^{jl} |j\rangle\langle i| \otimes |k\rangle\langle l|, \\ \hat{Q}^{\text{T}_B} &= \sum_{i,j,k,l} Q_{ik}^{jl} |i\rangle\langle j| \otimes (|k\rangle\langle l|)^T = \sum_{i,j,k,l} Q_{ik}^{jl} |i\rangle\langle j| \otimes |l\rangle\langle k|, \end{aligned} \quad (2.90)$$

Similarly, the partial traces over \mathcal{H}_A and \mathcal{H}_B are definable as

$$\begin{aligned} \text{tr}_A[\hat{Q}] &= \sum_m (\langle m| \otimes \hat{I}) \hat{Q} (|m\rangle \otimes \hat{I}) = \sum_{m,k,l} Q_{mk}^{ml} |k\rangle\langle l|, \\ \text{tr}_B[\hat{Q}] &= \sum_n (\hat{I} \otimes \langle n|) \hat{Q} (\hat{I} \otimes |n\rangle) = \sum_{n,i,j} Q_{in}^{jn} |i\rangle\langle j|, \end{aligned} \quad (2.91)$$

where $\{|m\rangle\}$ and $\{|n\rangle\}$ are orthonormal bases of \mathcal{H}_A and \mathcal{H}_B , respectively.

2.3.1 Schmidt decomposition

A useful theorem for expressing composite vectors is the *Schmidt decomposition*: given a bipartite vector $|\psi\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$, there exists sets of orthonormal states $\{|k_A\rangle\}$ and $\{|k_B\rangle\}$ for systems A and B , respectively, such that

$$|\psi\rangle = \sum_k \lambda_k |k_A\rangle \otimes |k_B\rangle. \quad (2.92)$$

Here, λ_k are non-negative real numbers, known as *Schmidt coefficients* that satisfy

$$\sum_k \lambda_k^2 = \langle \psi | \psi \rangle. \quad (2.93)$$

The number of (non-zero) Schmidt coefficients is called the *Schmidt number* of the vector $|\psi\rangle$, and the bases $\{|k_A\rangle\}$ and $\{|k_B\rangle\}$ are called the *Schmidt bases*.

The Schmidt decomposition (2.92) is simply a way of expressing a vector in the tensor product of two inner product spaces (such as Hilbert spaces), and our ability to do so is a very powerful result. For example, given a normalized bipartite vector $|\psi\rangle$ with decomposition (2.92), the reduced operators for each system can be calculated to be

$$\begin{aligned} \hat{\rho}^A &\equiv \text{tr}_B[|\psi\rangle\langle\psi|] = \sum_k \lambda_k^2 |k_A\rangle\langle k_A|, \\ \hat{\rho}^B &\equiv \text{tr}_A[|\psi\rangle\langle\psi|] = \sum_k \lambda_k^2 |k_B\rangle\langle k_B|. \end{aligned} \quad (2.94)$$

The fact that the eigenvalues of both of these operators are identical (simply equal to λ_k^2) is a significant result. This is because many of the important properties of operators are completely determined by their eigenvalues. Consequently, such properties of the reduced operators of a vector on a composite system will automatically be the same, regardless of the structure of the composite vector.

2.3.2 Multiple systems of arbitrary dimension

We now turn our attention to d -dimensional Hilbert spaces, which we denote as \mathcal{H}_d . On such spaces, there always exists a vector basis $\{|e_i\rangle\}_{i=1}^d$ with which any $|\psi\rangle \in \mathcal{H}_d$ can be expressed as

$$|\psi\rangle = \sum_{i=1}^d \psi_i |e_i\rangle. \quad (2.95)$$

If we instead have N such systems $\{|\psi_n\rangle\}_{n=1}^N$, each residing in the same Hilbert space $|\psi_n\rangle \in \mathcal{H}_d$, then their composition is

$$\begin{aligned} |\psi_1\rangle \otimes |\psi_2\rangle \otimes \dots \otimes |\psi_N\rangle &\equiv \bigotimes_{n=1}^N |\psi_n\rangle \\ &= \bigotimes_{n=1}^N \sum_{i=1}^d \psi_{i_n} |e_{i_n}\rangle \\ &= \sum_{\{i_n\}_{n=1}^N = 1}^d \psi_{i_1} \psi_{i_2} \dots \psi_{i_N} |e_{i_1}\rangle \otimes |e_{i_2}\rangle \otimes \dots \otimes |e_{i_N}\rangle \\ &= \sum_{i=1}^d \left(\prod_{n=1}^N \psi_{i_n} \right) \bigotimes_{n=1}^N |e_{i_n}\rangle \\ &= \sum_{i=1}^d \psi_{i_1, i_2, \dots, i_N} |e_{i_1, i_2, \dots, i_N}\rangle \end{aligned} \quad (2.96)$$

where ψ_{i_n} is the i_n -th component of the n -th state (with basis $\{|e_{i_n}\rangle\}_{i_n=1}^d \equiv \{|e_i\rangle\}_{i=1}^d$ for all n), i.e.,

$$|\psi_n\rangle = \sum_{i_n=1}^d \psi_{i_n} |e_{i_n}\rangle, \quad (2.97)$$

the amalgamated components are

$$\psi_{i_1, i_2, \dots, i_N} \equiv \prod_{n=1}^N \psi_{i_n} = \psi_{i_1} \psi_{i_2} \dots \psi_{i_N}, \quad (2.98)$$

and we introduced the composite vector

$$|e_{\{i_n\}}\rangle \equiv |e_{i_1, i_2, \dots, i_N}\rangle \equiv \bigotimes_{n=1}^N |e_{i_n}\rangle. \quad (2.99)$$

Since $\{|e_{i_n}\rangle\}_{i_n=1}^d$ is a basis for the Hilbert space \mathcal{H}_d , then accordingly the set of states

$$\begin{aligned} \left\{ |e_{\{i_n\}}\rangle : 1 \leq i_n \leq d, 1 \leq n \leq N \right\} &\equiv \left\{ \bigotimes_{n=1}^N |e_{i_n}\rangle \right\}_{\{i_n\}_{n=1}^N = 1}^d \\ &= \{ |e_{i_1}\rangle \otimes |e_{i_2}\rangle \otimes \dots \otimes |e_{i_N}\rangle \}_{\{i_n\}_{n=1}^N = 1}^d, \end{aligned} \quad (2.100)$$

makes a basis for the composite space

$$\mathcal{H}_d^{\otimes N} \equiv \bigotimes_{i=1}^N \mathcal{H}_d \equiv \underbrace{\mathcal{H}_d \otimes \mathcal{H}_d \otimes \dots \otimes \mathcal{H}_d}_{N \text{ times}}. \quad (2.101)$$

As this is N compositions of d -dimensional systems, there are a total of d^N basis states $|e_{\{i_n\}}\rangle$ which span $\mathcal{H}_d^{\otimes N}$. This d^N -dimensional composite space may therefore be written as \mathcal{H}_{d^N} , and we have the isomorphism

$$\mathcal{H}_{d^N} \cong \mathcal{H}_d^{\otimes N}. \quad (2.102)$$

In other words, a d^N -dimensional Hilbert space \mathcal{H}_{d^N} is decomposable into exactly N tensor products $\mathcal{H}_d^{\otimes N}$ of a d -dimensional Hilbert space \mathcal{H}_d . Importantly, this means any linear operator $\hat{A} \in \mathcal{H}_{d^N}$ may be embedded in $\mathcal{H}_d^{\otimes N}$ via a decomposition into tensor products of generalized Gell-Mann matrices (with the identity $\hat{\lambda}_0 \equiv \hat{I}_d$) (see *Qudits* (page 31)),

$$\hat{A} = \frac{1}{d^N} \sum_{\{i_n\}_{n=1}^N=0}^{d^2-1} A_{i_1, i_2, \dots, i_N} \hat{\lambda}_{i_1} \otimes \hat{\lambda}_{i_2} \otimes \dots \otimes \hat{\lambda}_{i_N} \quad (2.103)$$

where the coefficients

$$A_{i_1, i_2, \dots, i_N} = \text{tr}[(\hat{\lambda}_{i_1} \otimes \hat{\lambda}_{i_2} \otimes \dots \otimes \hat{\lambda}_{i_N}) \hat{A}] \quad (2.104)$$

collectively characterize \hat{A} under this decomposition. For example, a bipartite operator on d -dimensional subsystems can be written as

$$\hat{A} = \frac{1}{d^2} \sum_{i_1, i_2=0}^{d^2-1} A_{i_1, i_2} \hat{\lambda}_{i_1} \otimes \hat{\lambda}_{i_2} \quad (2.105)$$

with $A_{i_1, i_2} = \text{tr}[(\hat{\lambda}_{i_1} \otimes \hat{\lambda}_{i_2}) \hat{A}]$. Alternatively, in the case of binary ($d = 2$) systems (see *Qubits* (page 29)), this decomposition (2.103) is accomplished using the Pauli matrices (3.16) (with the identity $\hat{\sigma}_0 \equiv \hat{I}_2$),

$$\hat{A} = \frac{1}{2^N} \sum_{\{i_n\}_{n=1}^N=0}^3 A_{i_1, i_2, \dots, i_N} \hat{\sigma}_{i_1} \otimes \hat{\sigma}_{i_2} \otimes \dots \otimes \hat{\sigma}_{i_N} \quad (2.106)$$

where

$$A_{i_1, i_2, \dots, i_N} = \text{tr}[(\hat{\sigma}_{i_1} \otimes \hat{\sigma}_{i_2} \otimes \dots \otimes \hat{\sigma}_{i_N}) \hat{A}]. \quad (2.107)$$

2.4 Superoperators

A *superoperator* is a linear map \mathbf{E} on the space of (linear) operators $\mathcal{L}(\mathcal{H})$,

$$\mathbf{E}[\cdot] : \mathcal{L}(\mathcal{H}) \rightarrow \mathcal{L}(\mathcal{H}), \quad (2.108)$$

which preserves linear combinations of operators, that is,

$$\mathbf{E}[\mu \hat{A} + \nu \hat{B}] = \mu \mathbf{E}[\hat{A}] + \nu \mathbf{E}[\hat{B}] \quad (2.109)$$

for all $\hat{A}, \hat{B} \in \mathcal{L}(\mathcal{H})$ and $\mu, \nu \in \mathbb{C}$. Any such superoperator may satisfy any number of additional properties:

- (i) (positivity): $\mathbf{E}[\hat{P}]$ is positive if \hat{P} is positive,
- (ii) (complete positivity): $\mathbf{E}[\hat{P}] \otimes \hat{I}$ is positive if $\hat{P} \otimes \hat{I}$ is positive,
- (iii) (Hermiticity preservation): $\mathbf{E}[\hat{H}]^\dagger = \mathbf{E}[\hat{H}]$ if $\hat{H}^\dagger = \hat{H}$,
- (iv) (trace preservation): $\text{tr}[\mathbf{E}[\hat{A}]] = \text{tr}[\hat{A}] \quad \forall \hat{A} \in \mathcal{L}(\mathcal{H})$.

2.5 Concavity and convexity

A Hilbert space \mathcal{H} is said to be *convex* if for any finite collection of its elements $\{\hat{A}_i\}$, the linear combination satisfies

$$\sum_i p_i \hat{A}_i \in \mathcal{H} \quad (2.111)$$

for any probability distribution $\mathbb{P} = \{p_i\}$. If this does not hold, then \mathcal{H} is said to be *concave*. An element $\hat{A}_i \in \mathcal{H}$ in a convex space is called *extreme* if it cannot be represented as a non-trivial combination of other elements (that is, it can only be expressed with a single non-zero p_i).

A (real) map

$$f(\cdot) : \mathcal{H} \rightarrow \mathbb{R} \quad (2.112)$$

is said to be convex if

$$f\left(\sum_i p_i \hat{A}_i\right) \leq \sum_i p_i f(\hat{A}_i) \quad (2.113)$$

and concave if

$$f\left(\sum_i p_i \hat{A}_i\right) > \sum_i p_i f(\hat{A}_i). \quad (2.114)$$

Quantum mechanics on discrete Hilbert spaces

The theory of *quantum mechanics* seeks to explain the dynamics of the particles of which our universe is composed. It differs from classical physics in that certain properties (such as energy and momentum) of bound systems are restricted to being within sets of discrete values—a characteristic known as *quantization*. Additionally, objects described by quantum mechanics have properties distinctive of both waves and particles (termed *wave-particle duality*), and there are fundamental limits to the accuracy with which values of specific physical quantities can be predicted prior to their measurement (described by the *uncertainty principle*). The theory presented here is based on the treatments in [1, 2, 4, 12, 13, 14, 15, 16, 17, 18, 19].

3.1 Quantum states

In a formal setup, any system in the discrete theory of quantum mechanics is a collection of identifiable physical properties at a given time. This is represented by a Hilbert space, which may be either infinite- or finite-dimensional. A usual presentation of that Hilbert space is a special function space, such as ℓ^2 or \mathbb{C}^d .

A *quantum state* is then simply a particular value (or, more properly, a probabilistic assignment) of these properties. It is this object which provides the probability distribution for the outcomes of each possible measurement on a system. Formally, we say that a quantum state represents (or is characterized by) a statistical ensemble $\mathbb{P} = \{p_i\}_i$ of independent, identically prepared copies of a quantum system. This is represented in general by a positive semi-definite Hermitian operator $\hat{\rho}$ of trace one (that is, a density operator) on the (discrete) Hilbert space \mathcal{H} . As the space of all density operators on \mathcal{H} is convex, then subsequently any convex combination of such operators is also a density operator, which allows us to express any $\hat{\rho}$ as

$$\hat{\rho} = \sum_i p_i \hat{\rho}_i. \quad (3.1)$$

This describes a mixture of the density operators $\{\hat{\rho}_i\}_i$ corresponding to the statistical ensemble \mathbb{P} . According to the spectral theorem (2.63), we are able to express any quantum state $\hat{\rho}$ in a d -dimensional Hilbert space \mathcal{H} equivalently as a convex combination of at most d vectors $|\psi_i\rangle$, i.e.,

$$\hat{\rho} = \sum_i p_i |\psi_i\rangle\langle\psi_i| \quad (3.2)$$

where the coefficients p_i collectively characterize $\hat{\rho}$.

3.1.1 Pure states

An extreme point in the convex space spanned by operators (3.1) is called a *pure state*, which we can denote in general as the outer product

$$\hat{\rho} = |\psi\rangle\langle\psi|. \quad (3.3)$$

This is necessarily a projection operator, i.e., $\hat{\rho}^2 = \hat{\rho}$. Due to this form, any given pure state can be completely (and uniquely) represented by the vector

$$|\psi\rangle = \sum_i c_i |e_i\rangle \quad (3.4)$$

where $\{|e_i\rangle\}_i$ is an orthonormal basis for the associated Hilbert space, and $c_i \in \mathbb{C}$ are the state-characterizing probability amplitudes.

The set of physical pure states $\mathcal{P}(\mathcal{H})$ on a Hilbert space \mathcal{H} is the set of all normalized (unit) vectors on \mathcal{H} , with vectors equal up to a global phase considered to be equivalent (as such global phase differences are not observable, i.e., constitute measurable quantities). Compactly, we write

$$\mathcal{P}(\mathcal{H}) \equiv \{|\psi\rangle \in \mathcal{H} : |||\psi\rangle||_2 = 1, |\psi\rangle \cong e^{i\theta}|\psi\rangle \forall \theta \in \mathbb{R}\}. \quad (3.5)$$

Note:

The concept of vectors which differ only up to a multiplicative constant of magnitude 1 (i.e., a phase factor $e^{i\theta}$) being identified is a direct consequence of the fact that such states are physically indistinguishable—that is, there is no performable experiment which is able to differentiate them. This leads to the idea that the notion of a pure “state” in quantum mechanics is perhaps better described by a more complete (and unique) object called a *ray*, residing in a projective Hilbert space. Put simply, a ray $\underline{\psi}$ is the set of all state vectors ψ which are scalar non-zero multiples of each other, i.e.,

$$\underline{\psi} = \{e^{i\theta}\psi : \psi \in \mathcal{H}, ||\psi||_2 = 1, \theta \in \mathbb{R}\}. \quad (3.6)$$

The vectors in this set are all physically equivalent and called *representants* of $\underline{\psi}$. For example, if $\psi \in \underline{\psi}$, then ϕ is also a representant of $\underline{\psi}$ if and only if $\phi = e^{i\theta}\psi$ for some $\theta \in \mathbb{R}$, and therefore corresponds to the same physical state. In fact, in this formalism, a (pure) “quantum state” is more properly identified as the ray itself, not just a (single) vector. Given a Hilbert space \mathcal{H} , any two $\psi_1, \psi_2 \in \mathcal{H}$ satisfy the equivalence relation $\psi_1 \sim \psi_2$ if and only if $\psi_1, \psi_2 \in \underline{\psi}$. As such, the rays of \mathcal{H} form the equivalence classes of this equivalence equation, which correspond physically to distinguishable quantum states. Thus, when we (imprecisely) describe a quantum state using a vector $|\psi\rangle$ in the context of quantum mechanics, we systematically use any one element of the corresponding ray as its representant, albeit with the specific choice being completely inconsequential to any predictions of the theory.

3.1.2 Mixed states

States which are not pure are termed *mixed*, and cannot be expressed in the form (3.3). A mixed state characterized by a uniform probability distribution \mathbb{P} is called *maximally mixed* and takes the form

$$\hat{\rho} = \frac{1}{d} \hat{I}_d \quad (3.7)$$

where \hat{I}_d is the $d \times d$ identity operator. It is useful to think of linear combinations such as that in (3.2) as “classical” superpositions in the sense that they involve the correspondingly classical mixing of otherwise fully quantum-mechanical states $\hat{\rho}_i = |\psi_i\rangle\langle\psi_i|$. In other words, a mixed quantum state is a *statistical ensemble* of pure states, and is also known simply as a *mixed ensemble*. Importantly, this contrasts with the physically distinct notion of *quantum superposition*, which is encapsulated by (3.4).

Unlike pure states, mixed states cannot be represented as a state vector (e.g., $|\psi\rangle$). Both objects however may be described as a density operator (e.g., $\hat{\rho}$), which serves as a generalization of the concept of state vectors (wave functions). The set of physical density operators (including both pure and mixed states) on a Hilbert space \mathcal{H} is the set of all Hermitian trace-one (linear) operators on \mathcal{H} , and we denote this as

$$\mathcal{D}(\mathcal{H}) \equiv \{\hat{\rho} \in \mathcal{L}(\mathcal{H}) : \hat{\rho}^\dagger = \hat{\rho}, \text{tr}[\hat{\rho}] = 1\}. \quad (3.8)$$

Density operators representing mixed (impure) quantum states arise in quantum mechanics in two distinct situations. The first of these is when the preparation of a system is not fully known. In this case, the (incomplete/limited) knowledge of the quantum state can be captured only by a density operator that describes a statistical (i.e., mixture) of all possible preparations. The second situation is when one wants to describe a physical system that is entangled with another. Quantum entanglement theoretically prevents the existence of complete knowledge about subsystems in an entangled state, and so it is impossible to represent the state of any such subsystems as a pure state.

Note:

A useful concept in the formalism of quantum information theory is that of *purification*. This process refers to the fact that any mixed state in a finite-dimensional Hilbert space can be viewed as the reduced state of some pure state. To see this, let $\hat{\rho} \in \mathcal{D}(\mathcal{H}_A)$ be a density matrix where $\dim(\mathcal{H}_A)$ is finite. Given this, it is always possible to introduce an additional Hilbert space \mathcal{H}_B , corresponding to a fictitious system often called a *reference* system (and has no direct physical significance), alongside a pure state $|\psi\rangle \in \mathcal{P}(\mathcal{H}_A \otimes \mathcal{H}_B)$ such that

$$\text{tr}_B[|\psi\rangle\langle\psi|] = \hat{\rho}. \quad (3.9)$$

We say that $|\psi\rangle$ *purifies* $\hat{\rho}$, which is a consequence that is only made possible by the fact that we have access to a larger (composite) Hilbert space.

3.1.3 Qubits

Many physical applications of this theory in quantum mechanics involve Boolean (i.e., binary, or two-valued) logic, which is describable completely by a pair of orthonormal vectors that are often labelled $|0\rangle$ and $|1\rangle$. The representations in \mathbb{C}^2 of these vectors is typically the pair

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (3.10)$$

Together, these form a computational basis that is canonically known as the *z*-basis (and is referred to by many simply as the *computational* basis), given that each is an eigenvector of the Pauli-Z operator $\hat{\sigma}_z$ (3.16). Any quantum state with such binary dimensionality is called a *qubit* (though this nomenclature is often used solely in the context of *pure* binary states), and describes a two-level system that forms the basic unit of quantum information (analogous to the bit in classical computing). In the computational basis $\{|0\rangle, |1\rangle\}$, the simplest qubit which we can express is the quantum superposition

$$|\phi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad |\alpha|^2 + |\beta|^2 = 1, \quad \alpha, \beta \in \mathbb{C}. \quad (3.11)$$

In the \mathbb{C}^2 -representation (3.10), this state can be expressed as

$$|\phi\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}. \quad (3.12)$$

An alternative parametrisation of the qubit vector state (3.11) takes the form

$$|\phi\rangle = e^{i\varsigma} \left[\cos\left(\frac{\vartheta}{2}\right)|0\rangle + e^{i\varphi} \sin\left(\frac{\vartheta}{2}\right)|1\rangle \right], \quad \varsigma, \vartheta, \varphi \in \mathbb{R}. \quad (3.13)$$

Given that the factor $e^{i\varsigma}$ represents a global phase, then all states with fixed ϑ, φ but different ς are physically equivalent, and so this factor can be disregarded. We are therefore left with a parametrization of a qubit wherein the numbers ϑ and φ define a point on the surface of a unit sphere, known as the *Bloch sphere* (illustrated in Figure 3.1). This is essentially a unique mapping of the 2-dimensional Hilbert space to the 3-dimensional real space \mathbb{R}^3 (with basis vectors $\{\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2, \hat{\mathbf{e}}_3\}$), which is accomplished by choosing the antipodal points on unit sphere in \mathbb{R}^3 to correspond to the pair of mutually orthogonal basis vectors $|0\rangle$ and $|1\rangle$. In this sense, the qubit state $\mathbf{r}_\phi \in \mathbb{R}^3$, which corresponds to its Hilbert space counterpart (3.11), may be written exactly as

$$\mathbf{r}_\phi = \sin(\vartheta) \cos(\varphi) \hat{\mathbf{e}}_1 + \sin(\vartheta) \sin(\varphi) \hat{\mathbf{e}}_2 + \cos(\vartheta) \hat{\mathbf{e}}_3. \quad (3.14)$$

Similarly, a 2-dimensional density operator $\hat{\rho}$ in the same representation corresponds to the vector

$$\mathbf{r}_\rho = \text{tr}[\hat{\rho}\hat{\sigma}_1]\hat{\mathbf{e}}_1 + \text{tr}[\hat{\rho}\hat{\sigma}_2]\hat{\mathbf{e}}_2 + \text{tr}[\hat{\rho}\hat{\sigma}_3]\hat{\mathbf{e}}_3 \quad (3.15)$$

where

$$\begin{aligned}\hat{\sigma}_1 = \hat{\sigma}_x &\equiv |0\rangle\langle 1| + |1\rangle\langle 0| = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \\ \hat{\sigma}_2 = \hat{\sigma}_y &\equiv -i|0\rangle\langle 1| + i|1\rangle\langle 0| = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \\ \hat{\sigma}_3 = \hat{\sigma}_z &\equiv |0\rangle\langle 0| - |1\rangle\langle 1| = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},\end{aligned}\quad (3.16)$$

are the *Pauli matrices*.

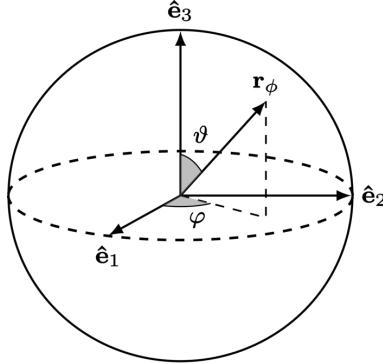


Figure 3.1: The Bloch sphere.

The Pauli matrices $\{\hat{\sigma}_k\}_{k=1}^3$ are Hermitian, and so represent observables in the context of quantum mechanics. Together with the identity operator (writing $\hat{\sigma}_0 = \hat{I}_2$), the set $\{\hat{\sigma}_\mu\}_{\mu=0}^3$ forms a complete basis for the space of complex 2×2 matrices. Since this coincides with the space of linear operators $\mathcal{L}(\mathcal{H})$ on a qubit Hilbert space \mathcal{H} , then any (density) operator $\hat{\rho} \in \mathcal{L}(\mathcal{H})$ can be expressed as the linear combination

$$\hat{\rho} = \frac{1}{2} \sum_{\mu=0}^3 \text{tr}[\hat{\sigma}_\mu \hat{\rho}] \hat{\sigma}_\mu. \quad (3.17)$$

In the context of density operators, since the coefficient for $\mu = 0$ in (3.17) is fixed by normalization (i.e., $\text{tr}[\hat{\sigma}_0 \hat{\rho}] = 1$), then the state $\hat{\rho}$ is completely (and uniquely) determined by the *Bloch vector*, which is defined, for an d -dimensional state, as the $(d^2 - 1)$ -dimensional real vector of parameters $\text{tr}[\hat{\sigma}_k \hat{\rho}]$ (for $k \geq 1$). Accordingly, the linear combination (3.17) is said to be the *Bloch sphere representation* of the state $\hat{\rho}$. Note also of course that any appropriate basis, not just the Pauli basis (with identity), allows for reconstruction of any qubit state in the manner of (3.17).

The Bloch vector provides a complete description of a quantum system because it represents the expectation values of specific informationally complete observables. In general, a complex $d \times d$ density matrix, which represents the state of an d -dimensional quantum system, is geometrically equivalent to a vector in an $(d^2 - 1)$ -dimensional real space. This is because the space of density matrices is a (bounded) convex subset of \mathbb{R}^{d^2} , with its elements possessing certain properties (namely Hermiticity, positive semi-definiteness, and trace equal to 1). Since this real space coincides with that of the (generalized) *Bloch ball*, including both the surface (the *Bloch sphere*, on which points correspond to pure states) and interior (in which points correspond to mixed states), then this is why determination of the Bloch vector is equivalent to identification of the quantum state. Therefore, as the Pauli operators form a set of informationally complete observables for qubit systems, then the inclusion of the identity operator to this set yields a *tomographically complete* basis for the space of 2×2 Hermitian matrices, of which the space of 2 -dimensional density matrices is a subset.

3.1.4 Qutrits

Quantum states realized by 3-dimensional systems are commonly known as *qutrits*. These objects are analogous to the “trit” unit of information in classical computing and so are useful for performing

calculations with ternary logic. Such systems are typically described using a set of orthonormal (basis) vectors 3-dimensional $\{|0\rangle, |1\rangle, |2\rangle\}$, with which a qutrit can in general be expressed as

$$|\phi\rangle = \alpha|0\rangle + \beta|1\rangle + \gamma|2\rangle, \quad |\alpha|^2 + |\beta|^2 + |\gamma|^2 = 1, \quad \alpha, \beta, \gamma \in \mathbb{C}. \quad (3.18)$$

In the \mathbb{C}^3 -representation, the basis vectors can take the form

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad |2\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad (3.19)$$

with which (3.18) can be rewritten as

$$|\phi\rangle = \alpha \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \gamma \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix}. \quad (3.20)$$

Qutrits described by 3-dimensional density operators have corresponding representations as 3×3 density matrices. One way of expressing such a state is via the linear combination

$$\hat{\rho} = \frac{1}{3} \sum_{\mu=0}^8 \text{tr}[\hat{\lambda}_\mu \hat{\rho}] \hat{\lambda}_\mu. \quad (3.21)$$

where $\{\hat{\sigma}_\mu\}_{\mu=0}^3$ is a complete basis for the space of 3×3 complex matrices. Perhaps the most notable choice of matrices with which to construct a such a basis are the *Gell-Mann matrices*,

$$\begin{aligned} \hat{\lambda}_1 &\equiv |0\rangle\langle 1| + |1\rangle\langle 0| &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\ \hat{\lambda}_2 &\equiv -i|0\rangle\langle 1| + i|1\rangle\langle 0| &= \begin{bmatrix} 0 & -i & 0 \\ i & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\ \hat{\lambda}_3 &\equiv |0\rangle\langle 0| - |1\rangle\langle 1| &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\ \hat{\lambda}_4 &\equiv |0\rangle\langle 2| + |2\rangle\langle 0| &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \\ \hat{\lambda}_5 &\equiv -i|0\rangle\langle 2| + i|2\rangle\langle 0| &= \begin{bmatrix} 0 & 0 & -i \\ 0 & 0 & 0 \\ i & 0 & 0 \end{bmatrix}, \\ \hat{\lambda}_6 &\equiv |2\rangle\langle 3| + |3\rangle\langle 2| &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \\ \hat{\lambda}_7 &\equiv -i|2\rangle\langle 3| + i|3\rangle\langle 2| &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -i \\ 0 & i & 0 \end{bmatrix}, \\ \hat{\lambda}_8 &\equiv \frac{1}{\sqrt{3}}(|0\rangle\langle 0| + |1\rangle\langle 1| - 2|2\rangle\langle 2|) &= \frac{1}{\sqrt{3}} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{bmatrix}, \end{aligned} \quad (3.22)$$

in addition to $\hat{\lambda}_0 \equiv \hat{I}_3$.

3.1.5 Qudits

Quantum vector states in a d -dimensional Hilbert space \mathcal{H}_d can in general be expressed as the superposition

$$|\phi\rangle = \sum_{i=0}^{d-1} c_i |i\rangle, \quad \sum_{i=0}^{d-1} |c_i|^2 = 1, \quad c_i \in \mathbb{C}. \quad (3.23)$$

which is often referred to as a *qudit*. Here, $\{|i\rangle\}_{i=0}^{d-1}$ is a vector basis for \mathcal{H}_d , and may be expressed in a \mathbb{C}^d -representation using the vectors in (2.37). With this, we can write (3.23) as

$$|\phi\rangle = c_0 \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + c_1 \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} + \dots + c_{d-1} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{d-1} \end{bmatrix}. \quad (3.24)$$

Just like in the cases of qubits and qutrits, general d -dimensional states described by density operators can be expressed via a linear combination of the form

$$\hat{\rho} = \frac{1}{d} \sum_{\mu=0}^{d^2-1} \text{tr}[\hat{\lambda}_\mu \hat{\rho}] \hat{\lambda}_\mu. \quad (3.25)$$

In order for this to be true, the matrices $\{\hat{\lambda}_\mu\}_{\mu=0}^{d^2-1}$ must form a basis for the space of $d \times d$ density matrices. Though there are multiple ways to find such a basis, doing so is a non-trivial task. Our construction here closely follows the work of [20], in which a suitable generalization of the Gell-Mann (or Pauli) matrices (3.22) is obtained.

We begin by introducing the elementary matrices $\{e_i^j\}_{i,j=1}^d$ which satisfy

$$(e_i^j)_{nm} = \delta_{ni}\delta_{mj}, \quad 1 \leq n \leq d, \quad 1 \leq m \leq d. \quad (3.26)$$

This defines a total of d^2 matrices, each of which possesses one entry equal to unity and all others equal to zero. Using these, we can construct $d(d - 1)$ traceless off-diagonal matrices,

$$\begin{aligned} \alpha_i^j &= e_i^j + e_j^i \\ \beta_i^j &= -i(e_i^j - e_j^i), \quad 1 \leq j < i \leq d \end{aligned} \quad (3.27)$$

and $d - 1$ traceless diagonal matrices,

$$\gamma_j^j = \sqrt{\frac{2}{j(j+1)}} \left[\sum_{k=1}^j e_k^k - j e_{j+1}^{j+1} \right]. \quad (3.28)$$

In total, we obtain $d^2 - 1$ traceless matrices. Note that these matrices are in fact the generators of $SU(d)$ (the group of $d \times d$ unitary matrices with determinant 1), meaning that they form a basis for the corresponding Lie algebra $\mathfrak{su}(d)$ (the space of $d \times d$ traceless Hermitian matrices). As the space of $d \times d$ Hermitian matrices has a real dimensionality of d^2 , then the difference between this space and $\mathfrak{su}(d)$ is a single degree of freedom that accounts for the ability of the matrices in the former to be non-traceless. This missing property can be incorporated into the latter by including the d -dimensional identity matrix \hat{I}_d into any of its valid bases. In other words, the set of matrices that consists of both the generators of $SU(d)$ and the identity \hat{I}_d necessarily forms a basis for the entire space of $d \times d$ Hermitian matrices (both traceless and non-traceless).

With this in mind, we can use (3.27) and (3.28) to assemble a set of matrices,

$$\begin{aligned} \hat{\lambda}_{(i-1)^2+2(j-1)} &\equiv \alpha_i^j, \\ \hat{\lambda}_{(i-1)^2+2j-1} &\equiv \beta_i^j, \\ \hat{\lambda}_{i^2-1} &\equiv \gamma_{i-1}^{i-1}. \end{aligned} \quad (3.29)$$

These matrices are known as the *generalized Gell-Mann matrices*. In conjunction with the identity matrix $\hat{\lambda}_0 \equiv \hat{I}_d$, they form a suitable basis for the space of d -dimensional Hermitian matrices, meaning that any density matrix can indeed be expressed as the linear combination (3.25) using this basis.

3.2 Quantum quantities

3.2.1 Purity

The mixedness of a state may be quantified by the *purity* function,

$$\gamma(\hat{\rho}) \equiv \text{tr}[\hat{\rho}^2], \quad (3.30)$$

which defines a measure on the degree by which a state deviates from being (completely) pure. The purity of normalized d -dimensional states is bounded from above by 1 (perfect purity) and below by $\frac{1}{d}$ (maximal mixing), e.g.,

$$\frac{1}{d} \leq \gamma(\hat{\rho}) \leq 1, \quad (3.31)$$

and so a state $\hat{\rho}$ is pure if and only if $\gamma(\hat{\rho}) = 1$ (else it has some degree of mixing).

3.2.2 Trace distance

The *trace distance* is a metric on the space of density matrices, thereby providing a measure of the “distance” (or distinguishability) between two states. It is a generalization of the *Kolmogorov distance* for classical probability distributions to quantum density operators, and is defined for any two such operators $\hat{\rho}$ and $\hat{\tau}$ as

$$D(\hat{\rho}, \hat{\tau}) \equiv \frac{1}{2} \text{tr} |\hat{\rho} - \hat{\tau}|. \quad (3.32)$$

Here, we defined $|\hat{A}| \equiv \sqrt{\hat{A}^\dagger \hat{A}}$ to be the absolute value of an operator \hat{A} . Being a metric, the trace distance between density operators is always non-negative, i.e.,

$$D(\hat{\rho}, \hat{\tau}) \geq 0, \quad (3.33)$$

with equality (with zero) if and only if $\hat{\rho} = \hat{\tau}$. It is also necessarily symmetric, e.g., $D(\hat{\rho}, \hat{\tau}) = D(\hat{\tau}, \hat{\rho})$, and satisfies the *triangle inequality*, e.g.,

$$D(\hat{\rho}, \hat{\tau}) \leq D(\hat{\rho}, \hat{\sigma}) + D(\hat{\sigma}, \hat{\tau}), \quad (3.34)$$

for all density operators $\hat{\rho}$, $\hat{\tau}$, and $\hat{\sigma}$. In the case of normalized quantum states, the trace distance is bounded from above by unity, e.g.,

$$0 \leq D(\hat{\rho}, \hat{\tau}) \leq 1. \quad (3.35)$$

3.2.3 Fidelity

Another measure between two quantum states is the *fidelity*, which characterizes their similarity or “closeness”. It is defined for two density operators $\hat{\rho}$ and $\hat{\tau}$ as

$$F(\hat{\rho}, \hat{\tau}) \equiv \left(\text{tr} \sqrt{\sqrt{\hat{\rho}} \hat{\tau} \sqrt{\hat{\rho}}} \right)^2, \quad (3.36)$$

where the square roots of $\hat{\rho}$ and $\sqrt{\hat{\rho}} \hat{\tau} \sqrt{\hat{\rho}}$ (both positive-semidefinite matrices) are well-defined by the spectral theorem (2.63). While the fidelity is not a suitable metric on the space of density matrices, it still provides many of the properties that are expected of a useful distance measure. For example, it is non-negative for any input, e.g.,

$$0 \leq F(\hat{\rho}, \hat{\tau}) \leq 1, \quad (3.37)$$

with equivalence to its upper bound of unity if and only if $\hat{\rho} = \hat{\tau}$.

The fidelity expressed in (3.36) is the original, traditional form that is often cumbersome to use and inefficient to compute. Simpler yet equivalent reformulations are therefore desired, and perhaps the most encountered of such expressions is

$$F(\hat{\rho}, \hat{\tau}) \equiv \left(\text{tr} \left| \sqrt{\hat{\rho}} \sqrt{\hat{\tau}} \right| \right)^2, \quad (3.38)$$

where $|\hat{A}| \equiv \sqrt{\hat{A}^\dagger \hat{A}}$. Recent work [21, 22] has shown that, by leveraging the fact that the trace of a diagonalizable square matrix is equal to the sum of its eigenvalues, an even simpler expression for the fidelity can be written as

$$F(\hat{\rho}, \hat{\tau}) \equiv \left(\text{tr} \sqrt{\hat{\rho} \hat{\tau}} \right)^2, \quad (3.39)$$

which holds true regardless of whether $\hat{\rho}$ and $\hat{\tau}$ do or do not commute.

In the case where either density matrix is pure, we can write

$$F(\hat{\rho}, \hat{\tau}) = \text{tr}[\hat{\rho}\hat{\tau}] = \begin{cases} \langle\psi_\rho|\hat{\tau}|\psi_\rho\rangle & \text{when } \hat{\rho} = |\psi_\rho\rangle\langle\psi_\rho|; \\ \langle\psi_\tau|\hat{\rho}|\psi_\tau\rangle & \text{when } \hat{\tau} = |\psi_\tau\rangle\langle\psi_\tau|; \end{cases} \quad (3.40)$$

which further reduces to the overlap

$$F(\hat{\rho}, \hat{\tau}) = |\langle\psi_\rho|\psi_\tau\rangle|^2 \quad (3.41)$$

when both states are pure.

Lastly, note that an alternative definition of the fidelity exists that specifies the same expression as (3.36) except without the square, i.e.,

$$F(\hat{\rho}, \hat{\tau}) \equiv \text{tr}\sqrt{\sqrt{\hat{\rho}}\hat{\tau}\sqrt{\hat{\rho}}}. \quad (3.42)$$

This form, sometimes called the *root fidelity*, shares many of the same properties as (3.36), but is ostensibly less common in the literature.

3.2.4 Entropy

In statistical physics, the concept of *entropy* commonly provides a quantification of how much uncertainty there is in the state of a physical system. In other words, it characterizes the “amount” of mixedness of a quantum state (that is, the degree to which the state is mixed). For quantum systems, the standard measure of entropy is the *von Neumann entropy*, defined for an arbitrary state $\hat{\rho}$ as

$$S(\hat{\rho}) \equiv -\text{tr}[\hat{\rho} \log_b \hat{\rho}]. \quad (3.43)$$

This definition can be equivalently expressed as

$$S(\hat{\rho}) = -\sum_k \lambda_k \log_b \lambda_k \quad (3.44)$$

where λ_k denote the non-zero eigenvalues in the spectrum of $\hat{\rho}$. Here, the matrix logarithm is taken to base b , which is the dimensionality of the unit of information in which the entropy is measured. For example, if $b = 2$, then $S(\hat{\rho})$ will be measured in “bits”. Many definitions of the von Neumann entropy alternatively specify $b = e$, for which the entropy is measured in so-called “nats”.

The von Neumann entropy is the quantum counterpart of the *Shannon entropy* in classical information theory. For a pure state, it is always zero, while for a mixed state, it is strictly positive with an upper bound of $\log_b d$ (achieved with a maximally mixed state) in a unipartite d -dimensional Hilbert space. Mathematically, we can write

$$0 \leq S(\hat{\rho}) \leq \log_b d \quad (3.45)$$

with $S(\hat{\rho}) = 0$ if and only if $\hat{\rho}$ is pure.

The quantum *relative entropy* provides a measure of distinguishability between two quantum density matrices, and is defined as

$$\begin{aligned} S(\hat{\rho}\|\hat{\tau}) &\equiv -\text{tr}[\hat{\rho} \log_b \hat{\tau}] - S(\hat{\rho}) \\ &= \text{tr}[\hat{\rho}(\log_b \hat{\rho} - \log_b \hat{\tau})], \end{aligned} \quad (3.46)$$

where $\hat{\rho}$ and $\hat{\tau}$ are arbitrary quantum states of the same dimensionality. It is analogous to the notion of relative entropy from classical information theory, wherein it characterizes the “distance” between two probability distributions. *Klein’s inequality* states that the quantum relative entropy is non-negative, i.e.,

$$S(\hat{\rho}\|\hat{\tau}) \geq 0, \quad (3.47)$$

for any two density matrices $\hat{\rho}$ and $\hat{\tau}$, with equality if and only if $\hat{\rho} = \hat{\tau}$.

3.2.5 Mutual information

The quantum *mutual information* is a measure of correlation between the subsystems of a multipartite quantum system. Let $\hat{\rho}^{A,B}$ be a bipartite state for a composite system with Hilbert space $\mathcal{H}_A \otimes \mathcal{H}_B$, and let

$$\begin{aligned}\hat{\rho}^A &\equiv \text{tr}_B[\hat{\rho}^{A,B}], \\ \hat{\rho}^B &\equiv \text{tr}_A[\hat{\rho}^{A,B}],\end{aligned}\tag{3.48}$$

be the reduced states on subsystems A and B , respectively. Using these, we can define the mutual information between these subsystems as

$$I(A : B) \equiv S(\hat{\rho}^A) + S(\hat{\rho}^B) - S(\hat{\rho}^{A,B}),\tag{3.49}$$

where $S(\hat{\rho})$ is the von Neumann entropy of a state $\hat{\rho}$ as defined in (3.43). An equivalent expression uses the relative entropy (3.46) to write

$$I(A : B) = S(\hat{\rho}^{A,B} \parallel \hat{\rho}^A \otimes \hat{\rho}^B).\tag{3.50}$$

The quantum mutual information is analogous to the *Shannon mutual information* in classical information theory. It is a non-negative quantity, i.e.,

$$I(A : B) \geq 0,\tag{3.51}$$

with equality if and only if the subsystems are independent (and are thus uncorrelated).

3.3 Separability and entanglement

A vector state $|\psi\rangle \in \mathcal{P}(\mathcal{H}_0 \otimes \mathcal{H}_1 \otimes \dots \otimes \mathcal{H}_{N-1})$ over N subsystems is called *separable* or a *product state* if it can be written in the form

$$\begin{aligned}|\psi\rangle &= |\psi_0\rangle \otimes |\psi_1\rangle \otimes \dots \otimes |\psi_{N-1}\rangle \\ &= \bigotimes_{n=0}^{N-1} |\psi_n\rangle,\end{aligned}\tag{3.52}$$

where $|\psi_n\rangle \in \mathcal{P}(\mathcal{H}_n)$ is a vector state on the n -th subsystem. Similarly, a mixed state $\hat{\rho} \in \mathcal{D}(\mathcal{H}_0 \otimes \mathcal{H}_1 \otimes \dots \otimes \mathcal{H}_{N-1})$ over N subsystems is separable if it can be expressed as

$$\begin{aligned}\hat{\rho} &= \sum_k p_k \hat{\rho}_k^0 \otimes \hat{\rho}_k^1 \otimes \dots \otimes \hat{\rho}_k^{N-1} \\ &= \sum_k p_k \bigotimes_{n=0}^{N-1} \hat{\rho}_k^n,\end{aligned}\tag{3.53}$$

where $\hat{\rho}_k^n \in \mathcal{D}(\mathcal{H}_n)$ are density operators representing the n -th subsystem, and the coefficients p_k are non-negative real numbers that collectively satisfy $\sum_k p_k = 1$. If there exists only a single non-zero p_k , then the state is called *simply separable* or, similarly to the separable vector state case, a *product state*.

A quantum state that is not separable (*non-separable*) is said to be *entangled*. The corresponding phenomenon, *quantum entanglement*, is one of the most fascinating features of quantum mechanics. It manifests physically as quantum correlations (of various quantum properties such as observable quantities) between the individual subsystems of a composite quantum system. For example, a group of particles is entangled when the quantum state of each individual particle cannot be described independently of the state of the other particles in a collective description. This behaviour is realized to varying degrees in many different kinds of physical situations, including most interestingly when the particles are spacelike separated.

Quantum entanglement is also necessarily characterized by *unrealized* correlations, such that the outcomes of measurements on the quantum system(s) are completely undecided until the measurement occurs. This means that the very act of performing a measurement on one subsystem of an entangled

composition affects the entire system as a whole (i.e., apparent and irreversible collapse of the wave function). Though any such influence occurs seemingly instantaneously, quantum entanglement however does not allow any information to propagate superluminally (i.e., faster than the speed of light), despite the possibility that the subsystems of an entangled system are spacelike separated.

A quantum system is said to be *maximally entangled* if it cannot be separated into pure states on each of its subsystems. This can only occur between exactly two particles, which is to say that a single particle cannot be maximally entangled with more than one other particle at a time. This is a property called *monogamy*, and is a restriction that results in perfect quantum correlations between any two such entangled subsystems. For qubits, the maximally entangled states are the *Bell states*, of which there are four:

$$\begin{aligned} |\Phi^+\rangle &\equiv \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle), \\ |\Phi^-\rangle &\equiv \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle - |1\rangle \otimes |1\rangle), \\ |\Psi^+\rangle &\equiv \frac{1}{\sqrt{2}}(|0\rangle \otimes |1\rangle + |1\rangle \otimes |0\rangle), \\ |\Psi^-\rangle &\equiv \frac{1}{\sqrt{2}}(|0\rangle \otimes |1\rangle - |1\rangle \otimes |0\rangle). \end{aligned} \quad (3.54)$$

For systems composed of three or more subsystems (e.g., particles), any multipartite entanglement that manifests cannot be maximal as in the bipartite case (due to the monogamy), but still exists in many quantitatively different types. An example of an entangled three-qubit system is the *Greenberger-Horne-Zeilinger (GHZ) state*,

$$|\text{GHZ}\rangle = \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle \otimes |1\rangle). \quad (3.55)$$

Discarding (tracing out) any of its subsystems results in a separable state. Dissimilarly, for the three-qubit *W state*,

$$|\text{W}\rangle = \frac{1}{\sqrt{3}}(|0\rangle \otimes |0\rangle \otimes |1\rangle + |0\rangle \otimes |1\rangle \otimes |0\rangle + |1\rangle \otimes |0\rangle \otimes |0\rangle), \quad (3.56)$$

discarding any single subsystem yields a bipartite state that is still entangled.

3.4 Quantum operations

A *quantum operation* is a linear, trace-non-increasing, completely positive map

$$\mathbf{E} : \hat{\rho} \mapsto \mathbf{E}[\hat{\rho}] \in \mathcal{D}(\mathcal{H}). \quad (3.57)$$

It is convenient to express such maps using the general form

$$\mathbf{E}[\hat{\rho}] = \sum_i \hat{K}_i^\dagger \hat{\rho} \hat{K}_i \quad (3.58)$$

where $\mathbb{K} = \{\hat{K}_i\}$ is a set of linear operators, known as *Kraus operators*. The characteristic of *trace-non-increasing* is the requirement that $\text{tr}[\hat{\rho}] \geq \text{tr}[\mathbf{E}[\hat{\rho}]]$. Since it is (almost) always assumed that the input state $\hat{\rho}$ satisfies $\text{tr}[\hat{\rho}] = 1$, then the Kraus operators of a trace-non-increasing quantum operation in turn satisfy

$$\sum_i \hat{K}_i^\dagger \hat{K}_i \leq \hat{I}. \quad (3.59)$$

In the case of equivalence, the operation is said to be *complete* (and *trace-preserving*), otherwise it is *incomplete* (and *trace-decreasing*). An operation $\mathbf{E}[\hat{\rho}]$ is also said to be *physical* if it satisfies

$$0 \leq \text{tr}[\mathbf{E}[\hat{\rho}]] \leq 1. \quad (3.60)$$

Quantum operations are fundamental to an information-theoretic treatment of quantum mechanics as they provide the general basic formalism by which both state evolution and measurement can be described.

3.5 Quantum evolutions

In quantum mechanics, the term *evolution* refers to the transformation (either continuous or discrete) of quantum states over some parameter space (usually time). In general, there are two fundamentally distinct types of quantum evolution: those of which are *closed*, and those of which are *open*.

3.5.1 Closed quantum systems

In a closed quantum system, the evolution of a quantum state $\hat{\rho}$ is described simply by a unitary transformation \hat{U} of the state, which can be expressed as the quantum operation

$$\mathbf{E}[\hat{\rho}] = \hat{U}\hat{\rho}\hat{U}^\dagger. \quad (3.61)$$

This is naturally a trace-preserving map, that is, $\text{tr}[\mathbf{E}[\hat{\rho}]] = \text{tr}[\hat{\rho}]$, since \hat{U} is a unitary operator. In the specific case of a state that is initially a vector, we have

$$\mathbf{E}[\psi] = \hat{U}|\psi\rangle. \quad (3.62)$$

Perhaps the most prevailing example is the time evolution of the state $|\psi\rangle$, which is describable by a unitary of the form

$$\hat{U}(t) = \exp\left[-\frac{i}{\hbar}\hat{H}t\right] \quad (3.63)$$

where t is the time duration, \hat{H} is a Hermitian operator corresponding to the Hamiltonian of the system, and \hbar is the reduced Planck constant.

3.5.2 Open quantum systems

In an open quantum system, the evolution $\mathbf{E}[\hat{\rho}]$ of a quantum state $\hat{\rho}$ is described by the trace-preserving quantum operation

$$\mathbf{E}[\hat{\rho}] = \sum_i \hat{K}_i \hat{\rho} \hat{K}_i^\dagger \quad (3.64)$$

where the Kraus operators are given by

$$\hat{K}_i = \langle e_i | \hat{U} | \psi \rangle. \quad (3.65)$$

This form, which is sometimes called the *operator-sum representation*, describes the evolution of some system in the Hilbert space \mathcal{H}_S that interacts with an environment system (a separate system with Hilbert space \mathcal{H}_E) some unitary \hat{U} . In this bipartite setup, the environment is initially in the state $|\psi\rangle$ (with $\{|e_i\rangle\}$ denoting an orthonormal basis). Dropping the assumption that the environment is initially pure, we may in general write

$$\mathbf{E}[\hat{\rho}] = \text{tr}_E [\hat{U}(\hat{\rho}_S \otimes \hat{\tau}_E)\hat{U}^\dagger] \quad (3.66)$$

where $\hat{\tau}$ is the environment state. The output of the environment under this evolution is complementarily given by

$$\tilde{\mathbf{E}}[\hat{\tau}] = \text{tr}_S [\hat{U}(\hat{\rho}_S \otimes \hat{\tau}_E)\hat{U}^\dagger]. \quad (3.67)$$

Quantum operations of this kind, i.e., those which are both completely positive (CP) and trace-preserving (TP), are known as *CPTP maps* or, more popularly, *quantum channels*. Note however that there is no loss in generality in making the assumption that the environment is in a pure state, since it can always be purified by the introduction of an additional system.

3.6 Quantum measurements

A *quantum measurement* of a quantum state is expressible in general as the map

$$\mathbf{E}_i[\hat{\rho}] = \hat{K}_i \hat{\rho} \hat{K}_i^\dagger. \quad (3.68)$$

Here, $\{\hat{K}_i\}$ are Kraus operators which necessarily satisfy the completeness relation

$$\sum_i \hat{K}_i^\dagger \hat{K}_i = \hat{I} \quad (3.69)$$

since all possible measurement values together must necessarily form a complete quantum operation. Note that by writing the form

$$\hat{K}_i = \hat{U}_i \sqrt{\hat{M}_i} \quad (3.70)$$

where $\mathbb{U} = \{\hat{U}_i\}$ is a set of unitary operators and $\mathbb{M} = \{\hat{M}_i\}$ is a set of positive operators, we can, due to the relation $\hat{K}_i^\dagger \hat{K}_i = \hat{M}_i$ and the completeness of the Kraus operators, conclude that

$$\sum_i \hat{M}_i = \hat{I}. \quad (3.71)$$

The set \mathbb{M} is therefore called a *quantum observable*, or more formally, a *positive operator-valued measure* (POVM), and has an associated probability distribution $\mathbb{P} = \{p_i\}$ given by

$$p_i = \text{tr}[\hat{\rho} \hat{M}_i] = \text{tr}[\hat{K}_i \hat{\rho} \hat{K}_i^\dagger]. \quad (3.72)$$

Given that preservation of the trace cannot be guaranteed under the measurement, then accordingly the post-measurement state $\hat{\rho}'$ can be normalized if required, i.e.,

$$\hat{\rho}' = \frac{\mathbf{E}_i[\hat{\rho}]}{\text{tr}[\mathbf{E}_i[\hat{\rho}]]}. \quad (3.73)$$

This is to ensure that a physical post-measurement state is always recovered, which is to say that $\text{tr}[\hat{\rho}'] = 1$. Quantum measurement can therefore be thought as the extension of the concept of trace-non-preserving quantum operations to include normalization, the procedure of which makes it distinct from such ordinary quantum operations (being necessarily linear). In essence, a quantum measurement describes the probability of obtaining a particular outcome of a quantum operation on a system, in addition to the change in the state of that affected system. If however the result of the measurement is lost, then the post-measurement quantum state is (probabilistically) the sum of all possible outcomes (without renormalization) of the associated POVM,

$$\hat{\rho}' = \sum_i \hat{K}_i \hat{\rho} \hat{K}_i^\dagger. \quad (3.74)$$

This defines a linear, trace-preserving, completely positive map.

A quantum observable $\mathbb{M} = \{\hat{\Pi}_i\}$ is said to be *sharp* if, in addition to having positive operators, each operator $\hat{\Pi}_i$ is a projector, that is, $\hat{\Pi}_i = \hat{\Pi}_i^2$. According to the spectral theorem (2.63), every Hermitian operator \hat{H} may be expressed as

$$\hat{H} = \sum_i \lambda_i \hat{\Pi}_i \quad (3.75)$$

with eigenvalues λ_i . This means that the entire observable \mathbb{M} is representable by the single Hermitian operator \hat{H} , which is why such operators are typically referred to simply as *observables* in standard quantum theory. Given this expression, the probability of measuring an eigenvalue λ_i is given by

$$p_i = \text{tr}[\hat{\rho} \hat{\Pi}_i], \quad (3.76)$$

which is known as the *Born rule*, and is one of the key principles of quantum mechanics. Accordingly, the *expected value* of the associated observable is simply

$$\langle \hat{H} \rangle = \text{tr}[\hat{\rho} \hat{H}]. \quad (3.77)$$

3.7 Integrals over quantum states

Let \mathcal{H} denote the Hilbert space for a d -dimensional quantum system. The integral over quantum states $\hat{\rho} \in \mathcal{D}(\mathcal{H})$ is defined [14, 23] as

$$I = \int_{\mathcal{D}(\mathcal{H})} d[\hat{\rho}] J(\hat{\rho}) \quad (3.78)$$

where $J : \mathcal{D}(\mathcal{H}) \rightarrow \mathbb{C}$ is any scalar function, and $d[\hat{\rho}]$ is the integration measure over $\mathcal{D}(\mathcal{H})$. To compute the integral, an explicit form of the measure must be defined, and we are typically free to choose any such form in accordance with what we wish to accomplish. In the case of mixed states $\hat{\rho} \in \mathcal{D}(\mathcal{H})$, there is in fact no unique natural measure on $\mathcal{D}(\mathcal{H})$. One must therefore be chosen (along with a way to parametrize $\hat{\rho}$), which means that that is no unique, natural way to define the integral I , as it will depend on the choice of measure used.

For the specific case of pure states $|\phi\rangle \in \mathcal{P}(\mathcal{H})$ however, we are able to define both a suitable measure and parametrization. Given some $J : \mathcal{P}(\mathcal{H}) \rightarrow \mathbb{C}$, the integral over pure states [14, 23] is

$$I = \int_{\mathcal{P}(\mathcal{H})} d[\phi] J(\phi) \quad (3.79)$$

An important class of unitary transformations consists of random unitary matrices distributed according to the Haar measure on the group of $d \times d$ unitary matrices $U(d)$. This is significant for our purposes, as conveniently there exists a unique, natural measure over $\mathcal{P}(\mathcal{H})$ that is invariant under such transformations, and perhaps the best way to write this is in the Hurwitz parametrization [14, 24, 25]. Let us first introduce the Bloch sphere parametrization, often used for qubits, which involves the rotationally invariant area measure on a (unit) 2-sphere:

$$|\phi\rangle = \sin(\vartheta)|0\rangle + e^{i\varphi} \cos(\vartheta)|1\rangle, \quad (3.80)$$

$$d[\phi] \propto \sin(2\vartheta) d(2\vartheta) d\varphi.$$

The Hurwitz parametrization is then simply a generalization of this Bloch sphere parametrization to d -dimensional (pure) states. Therefore, given some orthonormal basis $\{|\mu\rangle\}_{\mu=0}^{d-1}$ of \mathcal{H} , any pure state $|\phi\rangle \in \mathcal{P}(\mathcal{H})$ may be expressed in this parametrization as

$$|\phi\rangle = \prod_{\nu=d-1}^1 \sin(\vartheta_\nu)|0\rangle + \sum_{\mu=1}^{d-2} e^{i\varphi_\mu} \cos(\vartheta_\mu) \prod_{\nu=d-1}^{\mu+1} \sin(\vartheta_\nu)|\mu\rangle + e^{i\varphi_{d-1}} \cos(\vartheta_{d-1})|d-1\rangle \quad (3.81)$$

for parameters $\vartheta_\mu \in [0, \frac{\pi}{2}]$ and $\varphi_\mu \in [0, 2\pi)$. The corresponding integration measure (unique up to a multiplicative constant) in this parametrization takes the form

$$d[\phi(\vartheta_\mu, \varphi_\mu)] = \prod_{\mu=1}^{d-1} \cos(\vartheta_\mu) \sin^{2\mu-1}(\vartheta_\mu) d\vartheta_\mu d\varphi_\mu, \quad (3.82)$$

which provides a natural measure for integrating over pure quantum states. Importantly, this is invariant under unitary operations, so that under $|\phi\rangle \rightarrow \hat{U}|\phi\rangle$, the measure transforms as $d[\phi] \rightarrow d[\hat{U}\phi] = d[\phi]$.

Quantum mechanics on continuous Hilbert spaces

One of the more physically meaningful classes of Hilbert spaces are those which are *continuous*. Such systems typically describe the dynamics of quantum particles propagating in some physical continuum, that is, \mathbb{R}^d for any $d \in \mathbb{Z}^+$. Mathematically, any continuous quantum system may be described by a state, which is a vector $|\psi\rangle$ residing in a Lebesgue space (a special function space that is also a type of Hilbert space), denoted $L^2(X)$, on certain set X which is some configuration space (such as \mathbb{R}^3). The theory discussed here adapts the work presented in [2, 4, 12, 26, 27, 28, 29, 30, 31, 32].

4.1 Probabilities and the Born rule

For a (measurable) function ψ , the condition $\psi \in L^2(X)$ specifies that ψ satisfies a finitely bound integral of the form

$$\|\psi\|_2 = \left[\int_X |\psi(\mathbf{x})|^2 d\mu(\mathbf{x}) \right]^{\frac{1}{2}} < \infty \quad (4.1)$$

where $\|\psi\|_2$ is the L^2 -norm of ψ and $\mu(\mathbf{x})$ is the appropriate measure of the set X . A function for which (4.1) holds true is said to be *square-integrable* (on X), or an L^2 -function. Note that such functions form an inner product space with the inner product given by

$$\langle \phi | \psi \rangle = \int_X \phi^*(\mathbf{x}) \psi(\mathbf{x}) d\mu(\mathbf{x}), \quad \phi, \psi \in L^2(\mathbf{x}). \quad (4.2)$$

With this, the Lebesgue space is now a Hilbert space, and we have the isomorphism $L^2 \cong \mathcal{H}$. From here, if the norm defined in (4.1) is equal to 1,

$$\int_X |\psi(\mathbf{x})|^2 d\mu(\mathbf{x}) = 1, \quad (4.3)$$

then the element $\psi \in L^2(X)$ defines a probability measure on X . In the case that the measure $\mu(\mathbf{x})$ is continuous, then the real-valued function

$$\mathbb{P}(\mathbf{x}) = |\psi(\mathbf{x})|^2 \quad (4.4)$$

is called a *probability density function* (PDF) and the complex-valued function $\psi(\mathbf{x})$ is called a *probability amplitude*. If the measure $\mu(\mathbf{x})$ is instead discrete, then the integral becomes a sum and $|\psi(\mathbf{x})|^2$ defines the probability measure on the set X —that is, the probability that the quantum system represented by $|\psi\rangle$ assumes the state with parameter $\mathbf{x} \in X$. The rule (4.4) is perhaps the archetypal representation of the Born rule.

The state vector $|\psi\rangle$ may be projected into different coordinate bases via the equivalence

$$\psi(\mathbf{x}) = \langle \mathbf{x} | \psi \rangle \iff |\psi\rangle = \int d\mu(\mathbf{x}) \psi(\mathbf{x}) | \mathbf{x} \rangle, \quad (4.5)$$

where the configuration of the associated system in d -dimensional space is represented by the d -tuple of generalized coordinates $(x_j) = \mathbf{x}$. Any such projection onto any basis is, by the usual interpretation of quantum mechanics, termed the *wave function* of the associated system. Commonly, this projection may be generalized to incorporate the state's time dependence,

$$\psi(\mathbf{x}, t) = \langle \mathbf{x}, t | \psi \rangle, \quad (4.6)$$

which we interpret as the probability amplitude that the associated system (e.g., a particle) has position \mathbf{x} at time t . Accordingly, the PDF is given by the Born rule (4.4),

$$\mathbb{P}(\mathbf{x}, t) = |\psi(\mathbf{x}, t)|^2, \quad (4.7)$$

which describes the probability of the particle's presence at and around the point (\mathbf{x}, t) .

4.2 Time evolution

Suppose we have some (closed) system which is described by the state $|\psi(t')\rangle$ defined on a complex Hilbert space at some initial time t' . By (3.62), the time evolution of this system over the interval $t' \rightarrow t''$ to some final state $|\psi(t'')\rangle$ is described by the form

$$|\psi(t'')\rangle = \hat{U}(t'', t')|\psi(t')\rangle. \quad (4.8)$$

Given that $\psi(t)$ must remain normalized for all t , then the operator \hat{U} must therefore be a unitary transformation. Importantly, this means that the transformation from past to future must be:

1. invertible \implies information is conserved,
2. norm-preserving \implies probability is conserved.

These facts imply that \hat{U} may be expressed in general as the exponential map

$$\hat{U}(t'', t') = \mathcal{T} \exp \left[-\frac{i}{\hbar} \int_{t'}^{t''} \hat{H}(t) dt \right] \quad (4.9)$$

where, given that the Lie algebra of the unitary group is generated by Hermitian operators, then \hat{H} is a Hermitian operator, i.e., $\hat{H}^\dagger = \hat{H}$. Here, the time-ordering symbol \mathcal{T} permutes a product of non-commuting operators \hat{A}_n into the uniquely determined reordered expression

$$\mathcal{T}[\hat{A}_1(t_1) \cdot \hat{A}_2(t_2) \cdot \dots \cdot \hat{A}_n(t_n)] = \hat{A}_{i_1}(t_{i_1}) \cdot \hat{A}_{i_2}(t_{i_2}) \cdot \dots \cdot \hat{A}_{i_n}(t_{i_n}), \quad t_{i_1} \geq t_{i_2} \geq \dots \geq t_{i_n}. \quad (4.10)$$

This result is a causal chain where the primary *cause* in the past is $\hat{A}_{i_n}(t_{i_n})$ while the future *effect* is $\hat{A}_{i_1}(t_{i_1})$. In the case where \hat{H} is time-independent, the exponential map is simply

$$\hat{U}(t'', t') = \exp \left[-\frac{i}{\hbar} (t'' - t') \hat{H} \right]. \quad (4.11)$$

Substituting this form into equation (4.8), letting $(t'', t') \rightarrow (t, 0)$ and then differentiating both sides with respect to t yields

$$\hat{H}|\psi(t)\rangle = i\hbar \frac{\partial}{\partial t} |\psi(t)\rangle. \quad (4.12)$$

This is the *time-dependent Schrödinger equation* (TDSE), which is perhaps one of the most fundamental and important equations of non-relativistic quantum mechanics. Here, we identify the operator \hat{H} as the (time-independent) *Hamiltonian* operator, which corresponds to (or, more formally, has eigenvalues of) the total energy E of the state ψ , i.e., $\hat{H}|\psi\rangle \equiv E|\psi\rangle$. Schrödinger's equation is a partial differential equation that governs the temporal evolution of any time-dependent quantum system described by the arbitrary state $|\psi(t)\rangle$. Essentially, it is the equation of motion which all non-relativistic systems must obey in order to be physical.

For any time-independent system, like a particle with (classical) kinetic energy T in a static potential field V , the Hamiltonian may be written as the sum of the (time-independent) total kinetic \hat{T} and potential \hat{V} (operator) energies,

$$\hat{H}(\mathbf{x}, \mathbf{p}) = \hat{T}(\mathbf{x}, \mathbf{p}) + \hat{V}(\mathbf{x}) \quad (4.13)$$

where $\mathbf{p} \equiv (p_j)$ represents the d -tuple of generalized momenta. Consequently, the TDSE is the mathematical statement that the time-independent Hamiltonian \hat{H} (4.13) is the infinitesimal generator of a one-parameter group parametrized by the time t . This is to say that the time evolution of a state $|\psi(t')\rangle$ to a later state $|\psi(t'')\rangle$ is generated by the Hamiltonian.

The time-evolution operator's utility can be observed in many different contexts. For example, consider an arbitrary single-body system located at position \mathbf{x} at time t . The state (or more specifically, the location) of such a system can be defined in the \mathbf{x} -space (configuration) basis as

$$|\mathbf{x}, t\rangle \equiv \hat{U}(t, 0)|\mathbf{x}\rangle, \quad (4.14)$$

so that we may write the overlap between the past $|\mathbf{x}', t'\rangle$ and future $|\mathbf{x}'', t''\rangle$ states as

$$\langle \mathbf{x}'', t'' | \mathbf{x}', t' \rangle = \langle \mathbf{x}'' | \hat{U}(t'', t') | \mathbf{x}' \rangle, \quad (4.15)$$

The physical significance of this quantity, known as the *transition amplitude*, shall next be discussed.

4.3 The path-integral formulation

The *path-integral formulation* [26, 33] (or the *sum-over-histories* approach) of quantum mechanics and field theory is a formalism that generalizes the principle of stationary action (from classical mechanics) to quantum mechanics. This formulation is based on the notion of the *propagator*,

$$K(\mathbf{x}'', t''; \mathbf{x}', t') \equiv \langle \mathbf{x}'', t'' | \mathbf{x}', t' \rangle, \quad t'' > t' \quad (4.16)$$

which is a generalized function, defined as per (4.15). Of its many properties and applications, a primary use of the propagator is in its ability to evolve a wave function $\psi(\mathbf{x}', t')$ to a later state via convolution, e.g.,

$$\psi(\mathbf{x}'', t'') = \int_{\mathbb{R}^d} K(\mathbf{x}'', t''; \mathbf{x}', t') \psi(\mathbf{x}', t') d^d \mathbf{x}'. \quad (4.17)$$

Here,

$$\psi(\mathbf{x}, t) \equiv \langle \mathbf{x}, t | \psi \rangle \quad (4.18)$$

is the wave function of the system in state $|\psi\rangle$ projected into \mathbf{x} -space at time t . Given its convolution property in (4.17) and its relation to the time-evolution operator in (4.15), we say that the propagator is the *Green's function* (or integral *kernel*) for the Schrödinger equation (4.12).

Similar to how a wave function can describe the probability amplitude for a particle's location \mathbf{x} at time t , we may say that the propagator $K(\mathbf{x}'', t''; \mathbf{x}', t')$ is the probability amplitude for the *transition* of the particle from (\mathbf{x}', t') to (\mathbf{x}'', t'') . In d -dimensional space, this transition could be accomplished through an infinite number of distinct and unique trajectories.

In general, the propagator can be written as the *phase-space path integral* over all paths $\mathbf{x}(t)$ and momentum evolutions $\mathbf{p}(t)$ from (\mathbf{x}', t') to (\mathbf{x}'', t'') ,

$$K(\mathbf{x}'', t''; \mathbf{x}', t') = \int_{\mathbf{x}'}^{\mathbf{x}''} \mathcal{D}\mathbf{x} \int_{\mathbf{p}'}^{\mathbf{p}''} \frac{\mathcal{D}\mathbf{p}}{(2\pi\hbar)^d} e^{iS[\mathbf{x}]/\hbar} \quad (4.19)$$

where $S[\mathbf{x}]$ is the action of path $\mathbf{x}(t)$. Here, we introduced the notation for the position and momentum differentials

$$\int_{\mathbf{x}_0}^{\mathbf{x}_N} \mathcal{D}\mathbf{x} \equiv \lim_{N \rightarrow \infty} \prod_{k=1}^{N-1} \int_{\mathbb{R}^d} d^d \mathbf{x}_k, \quad (4.20)$$

$$\int_{\mathbf{p}_0}^{\mathbf{p}_N} \mathcal{D}\mathbf{p} \equiv \lim_{N \rightarrow \infty} \prod_{n=0}^{N-1} \int_{\mathbb{R}^d} d^d \mathbf{p}_n, \quad (4.21)$$

which signify integration over all position $\mathbf{x}(t)$ and momentum $\mathbf{p}(t)$ evolutions through the N subdivisions of our phase space. This discretization arises from our partitioning of the time coordinate into N “slices”—physically, \mathbf{x}_n represents the position of the particle at time t_n (where $n \in \mathbb{Z}_1^N$)—and it is only after taking the continuum limit that we are able to recover smooth dynamics as described by (4.19).

For a system with a Hamiltonian whose dependence on momentum is purely quadratic, the integral over the momentum evolutions $\mathbf{p}(t)$ in (4.19) can be directly computed. Subsequently, the propagator reduces to the *configuration-space path integral*,

$$K(\mathbf{x}'', t''; \mathbf{x}', t') = \int_{\mathbf{x}'}^{\mathbf{x}''} \mathcal{D}'\mathbf{x} e^{iS[\mathbf{x}]/\hbar}, \quad (4.22)$$

where the “normalized” differential, compared to (4.20), is defined as

$$\int_{\mathbf{x}_0}^{\mathbf{x}_N} \mathcal{D}'\mathbf{x} \equiv \lim_{N \rightarrow \infty} \left[\frac{m}{2\pi i \hbar (t'' - t')} \right]^{\frac{N_d}{2}} \prod_{k=1}^{N-1} \int_{\mathbb{R}^d} d^d \mathbf{x}_k. \quad (4.23)$$

Here, m is the system's mass parameter. Note that this is an integral over *all* possible paths (both classical and quantum) and not just the classical path $\tilde{\mathbf{x}}(t)$ (that which satisfies the relevant classical equations of motion) with action $\tilde{S} \equiv S[\tilde{\mathbf{x}}]$. This expression is often presented as the typical form of the path integral.

5.1 Anatomy of a quantum circuit

Analogous to how the processing of information inside a classical computer may be represented diagrammatically by wires and logic gates, the *quantum circuitry* picturalism provides a way by which quantum operations can likewise be expressed visually. The formalism is based on the *Penrose graphical notation* (also often called *tensor network diagrams*)—a graphical notation for the visualization of tensors. This diagramming scheme is similar to logic circuit diagrams, in which wires depict the flow of information between logic gates (themselves representing logical operations on their input states).

A quantum circuit diagram is constructed using wires spanning input and output sections, upon which quantum gates are placed, describing manipulations and transformations (e.g., unitary operations) on the quantum states which follow these wires. A circuit's *inputs* (on the left) is a collection of one or more quantum states which, when encoded in an appropriate basis (for qubits this is typically the computational basis), is often called a (quantum) *register*. Oppositely, a circuit's *outputs* (on the right) is the set of quantum state(s) resulting from the transformations performed on the input(s) by the intermediary quantum gate(s). *Quantum computation* is then defined as any (unitary) evolution which takes an initial input state into some final output state. In this regard, it is important to note that, despite the “circuit” nomenclature, quantum circuits are typically not used to depict *closed loops* of state evolution.

One of the first uses of quantum circuit diagrams as we know them today was by Feynman [34]. The construction of these diagrams is described in more detail in [1, 13, 15, 17, 18]. All diagrams here were produced using the excellent LaTeX (TikZ) *Quantikz* package [35].

5.1.1 Fundamentals

- ▶ **Time:** Time advances horizontally from the left to the right. Each vertical slice of the circuit corresponds to a distinct instant of time in the temporal progression. In other words, vertically aligned events occur simultaneously.
- ▶ **Wires:** Each (physical) quantum or classical system (Hilbert space) is represented by a *wire* (also termed a *mode*) which is aligned with a single horizontal axis. Events at various points in time on these modes (such as state preparation, logical operations, measurement, etc.) are connected by lines (i.e., wires), thus forming a *circuit*.

- Quantum wires:

input quantum state ————— ...

... ————— output quantum state

- Classical wires:

input classical state ————— ...

... ————— output classical state

- ▶ **Omission:** Ellipses denote the omission of parts of the circuit. This notation is usually employed only in the context of the subsequent tutorial circuits as a way to give focus to the important parts of each example.
- ▶ **Ordering and labels:** For circuits with multiple systems, the order of tensor products (from left-to-right) in its corresponding mathematical expression matches the order of modes (from top-to-bottom) in the circuit. In the case of any ambiguity, the Hilbert spaces in which the states and operators reside are labelled using either superscripts or subscripts (depending on the context). If the systems are not explicitly named, the common convention is to denote them in order (from top-to-bottom) using the integers, starting from 0 (zero, as is customary in computer science) and counting up.

5.1.2 States

- ▶ State preparation or preselection is represented by:

$$\hat{\rho} \text{ ---} \iff \hat{\rho}$$

- ▶ The preparation of pure states is similarly:

$$|\phi\rangle \text{ ---} \iff |\phi\rangle\langle\phi|$$

- ▶ States prepared over several modes are denoted using a brace extending across all of the relevant subsystems:

$$\hat{\rho} \left\{ \text{---} \right. \iff \hat{\rho}^{0,1}$$

- ▶ The preparation of separable states is:

$$\begin{array}{c} \hat{\rho} \text{ ---} \\ |\phi\rangle \text{ ---} \end{array} \iff \hat{\rho} \otimes |\phi\rangle\langle\phi|$$

5.1.3 Gates

- ▶ Linear operators (usually unitary operators) acting on states are denoted by blocks called *gates*:

$$\dots \xrightarrow{\quad \hat{U} \quad} \iff \hat{U}(\dots)\hat{U}^\dagger$$

- ▶ Sequences of single-system operators are represented by sequences of gates:

$$\dots \xrightarrow{\quad \hat{A} \quad} \xrightarrow{\quad \hat{B} \quad} \iff \hat{B}\hat{A}(\dots)\hat{A}^\dagger\hat{B}^\dagger$$

- Operators which act on different subsystems within composite systems are represented distinctly by parallel gates:

$$\cdots \xrightarrow{\hat{A}} \cdots \quad \xleftrightarrow{\qquad} \quad (\hat{A} \otimes \hat{B})(\dots)(\hat{A}^\dagger \otimes \hat{B}^\dagger)$$

- An empty wire (i.e., the absence of a gate on a mode) corresponds to the identity operator:

$$\cdots \xrightarrow{\quad} = \cdots \xrightarrow{\hat{I}} \cdots \quad \xleftrightarrow{\qquad} \quad \hat{I}(\dots)\hat{I}^\dagger = (\dots)$$

- Groups of gates may be denoted with a bounding box, e.g.:

$$\xrightarrow{\hat{U}} \quad \xleftrightarrow{\qquad} \quad \hat{U} = \hat{B}\hat{A}$$

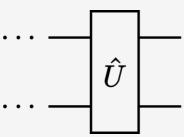
$$\xrightarrow{\hat{U}} \quad \xleftrightarrow{\qquad} \quad \hat{U} = \hat{A} \otimes \hat{B}$$

- Gates acting on different systems at distinct times (i.e., non-simultaneous operators) are separated horizontally. If all other wires in their instant are empty, then they may be combined:

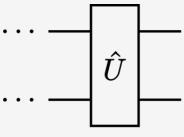
$$\xrightarrow{\hat{U}} \quad \xleftrightarrow{\qquad} \quad \begin{aligned} \hat{U} &= (\hat{I} \otimes \hat{B})(\hat{A} \otimes \hat{I}) \\ &= \hat{A} \otimes \hat{B} \end{aligned}$$

$$\xrightarrow{\hat{U}} \quad \xleftrightarrow{\qquad} \quad \begin{aligned} \hat{U} &= (\hat{A} \otimes \hat{I})(\hat{I} \otimes \hat{B}) \\ &= \hat{A} \otimes \hat{B} \end{aligned}$$

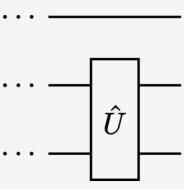
- Multipartite (composite) operators are denoted by gates which span the relevant wires:



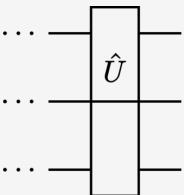
$$\iff \hat{U}^{0,1}(\dots)\hat{U}^{\dagger 0,1}$$



$$\iff (\hat{U}^{0,1} \otimes \hat{I}^2)(\dots)(\hat{U}^{\dagger 0,1} \otimes \hat{I}^2)$$



$$\iff (\hat{I}^0 \otimes \hat{U}^{1,2})(\dots)(\hat{I}^0 \otimes \hat{U}^{\dagger 1,2})$$

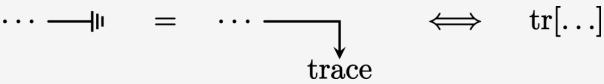


$$\iff \hat{U}^{0,2}(\dots)\hat{U}^{\dagger 0,2}$$

- If the system structure of the operators is unambiguous in the context of the circuit, then the (superscript) system labels are usually omitted for the sake of notational brevity.

5.1.4 Other operations

- The **trace** over a system (i.e., termination of a wire) is denoted by either a grounding symbol or a down arrow on the end of its corresponding mode:



- This extends neatly to the **partial trace**:



$$\cdots \xrightarrow{\parallel} \iff \text{tr}_{0,1}[\dots] = \text{tr}_0[\text{tr}_1[\dots]] = \text{tr}_1[\text{tr}_0[\dots]]$$

- **Postselection** against a state is simply the mirror of state preparation/preselection. This is indicated with either:

- The postselection state (explicitly in the dual Hilbert space, e.g., $\langle\phi|$ and $\hat{\rho}^\dagger$) at the termination of the wire.
- A round cap at the termination of the wire, followed by the postselection state.

$$\cdots \xrightarrow{\parallel} \langle\phi| = \cdots \xrightarrow{\parallel} \text{D} |\phi\rangle \iff \text{tr}[|\phi\rangle\langle\phi|(\dots)] = \langle\phi|(\dots)|\phi\rangle$$

$$\cdots \xrightarrow{\parallel} \hat{\rho}^\dagger = \cdots \xrightarrow{\parallel} \text{D} \hat{\rho} \iff \text{tr}[\hat{\rho}(\dots)]$$

$$\left. \begin{array}{c} \cdots \xrightarrow{\parallel} \\ \cdots \xrightarrow{\parallel} \end{array} \right\} \hat{\rho}^{\dagger 0,1} = \left. \begin{array}{c} \cdots \xrightarrow{\parallel} \text{D} \\ \cdots \xrightarrow{\parallel} \text{D} \end{array} \right\} \hat{\rho}^{0,1} \iff \text{tr}[\hat{\rho}^{0,1}(\dots)]$$

$$\left. \begin{array}{c} \cdots \xrightarrow{\parallel} \\ \cdots \xrightarrow{\parallel} \end{array} \right\} \hat{\rho}^\dagger = \left. \begin{array}{c} \cdots \xrightarrow{\parallel} \text{D} \hat{\rho} \\ \cdots \xrightarrow{\parallel} \end{array} \right\} \iff \text{tr}_1[(\hat{\rho} \otimes \hat{I})(\dots)]$$

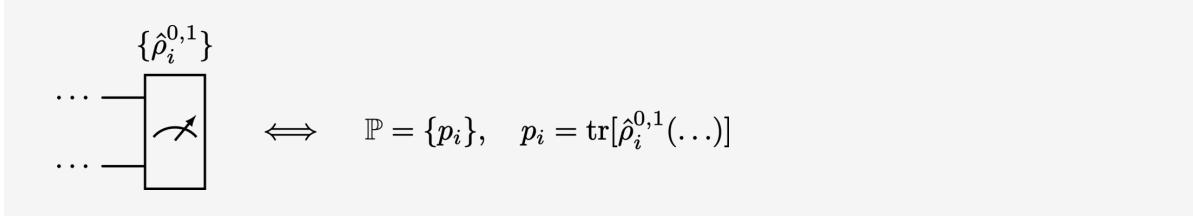
$$\left. \begin{array}{c} \cdots \xrightarrow{\parallel} \\ \cdots \xrightarrow{\parallel} \hat{\rho}^\dagger \end{array} \right\} = \left. \begin{array}{c} \cdots \xrightarrow{\parallel} \\ \cdots \xrightarrow{\parallel} \text{D} \hat{\rho} \end{array} \right\} \iff \text{tr}_1[(\hat{I} \otimes \hat{\rho})(\dots)]$$

$$\left. \begin{array}{c} \cdots \xrightarrow{\parallel} \\ \cdots \xrightarrow{\parallel} \hat{\rho}_0^\dagger \end{array} \right\} = \left. \begin{array}{c} \cdots \xrightarrow{\parallel} \text{D} \hat{\rho}_0 \\ \cdots \xrightarrow{\parallel} \hat{\rho}_1^\dagger \end{array} \right\} \iff \text{tr}_{0,1}[(\hat{\rho}_0 \otimes \hat{\rho}_1)(\dots)]$$

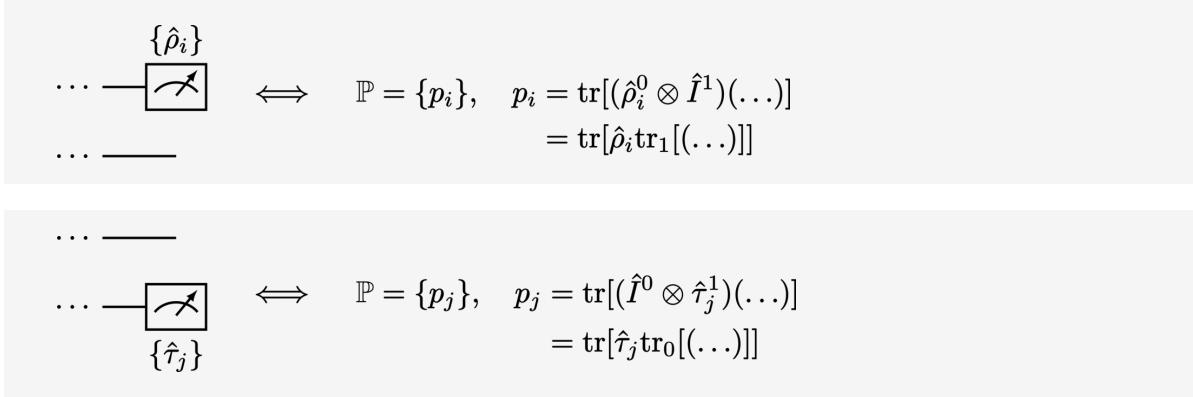
- Note that in most cases, the cap notation will be omitted for simplicity. This is particularly true when the postselection state is pre-defined (i.e., not unknown), and so the associated postselection is unambiguous.
- The **measurement** of a system with respect to a particular basis is represented by the termination of its wire with a meter:

$$\cdots \xrightarrow{\parallel} \boxed{\text{A}} \iff \mathbb{P} = \{p_i\}, \quad p_i = \langle\xi_i|(\dots)|\xi_i\rangle$$

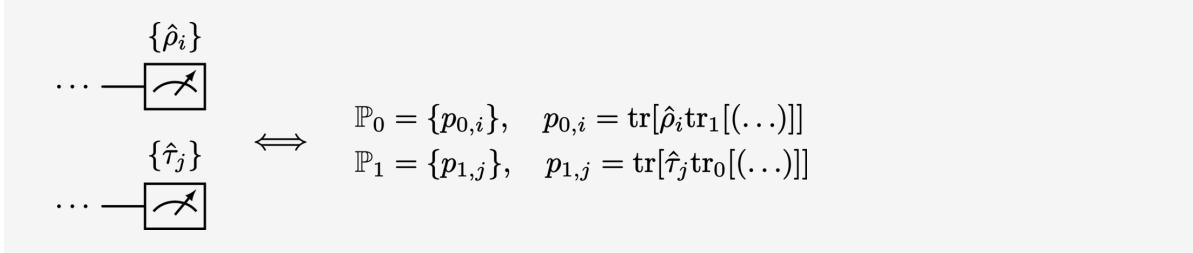
$$\cdots \xrightarrow{\parallel} \boxed{\text{A}} \iff \mathbb{P} = \{p_i\}, \quad p_i = \text{tr}[\hat{\rho}_i(\dots)]$$



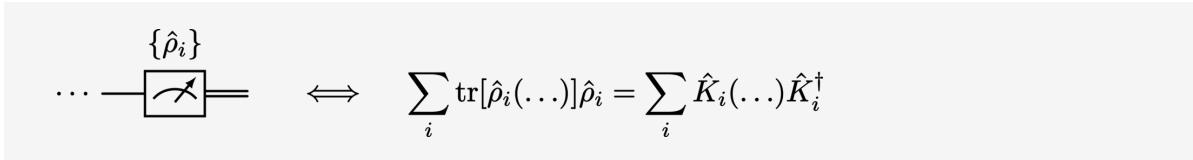
- ▶ Performing a measurement on a system returns a set of probabilities associated with the particular set of states against which the system was measured. In a circuit where measurement is not performed on every system, the measurement statistics are obtained by simply discarding (tracing out) all systems which do not undergo measurement:



- Note however that the output of the circuit still remains as the composite product of all unmeasured systems.
- ▶ Multiple measurements simply indicate that multiple sets of probabilities will be obtained, each corresponding to the independent measurement of their respective system, e.g.,



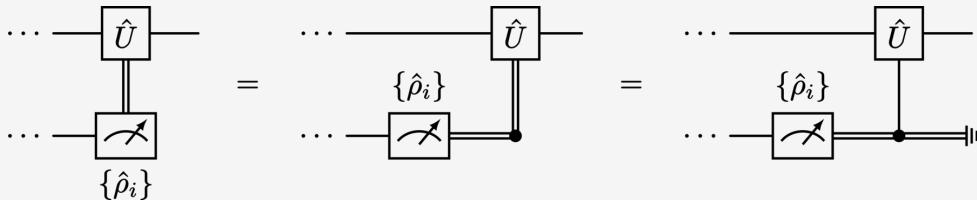
- ▶ A measurement for which the outcome is discarded (or forgotten) leaves the measured system in a superposition of all post-measurement states corresponding to all measurement operators $\{\hat{\rho}_i\}$. As this characteristically destroys purity, the post-gate wire of such a discarded measurement operation is visualized using a classical wire:



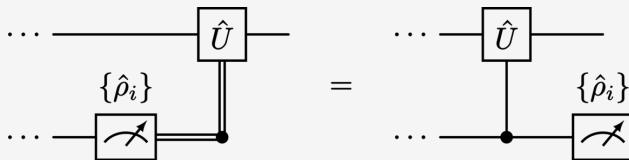
- ▶ Usage of the measurement outcomes as the (classical) control of a gate is denoted by the connection of the meter and the gate with a classical wire:

$$\begin{array}{ccc} \cdots & \xrightarrow{\hat{U}} & \cdots \\ & \downarrow & \\ \cdots & \xrightarrow{\{\hat{\rho}_i\}} & \end{array} \iff \sum_i \text{tr}_1[(\hat{U}_i \otimes \hat{\rho}_i)(\dots)(\hat{U}_i^\dagger \otimes \hat{I})] = \sum_i \hat{U}_i \text{tr}_1[(\hat{I} \otimes \hat{\rho}_i)(\dots)] \hat{U}_i^\dagger$$

- Note that a terminating post-measurement system like this is equivalent to a discarded measurement operation followed by a (partial) trace, and that a classical control wire is merely a device used to indicate the conveyance of classical information (and is in fact completely equivalent to a quantum control wire). This means we have the equivalence:



- The *principle of deferred measurement* states that measurement operations commute with control operations. Depicted visually, this is the equivalence:

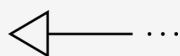


- The closed and timelike nature of the paths of chronology-violating systems (i.e., those which correspond to quantum states propagating along closed timelike curves) is captured by the use of triangular caps at the ends of the associated wires:

- Future connection:



- Past connection:



- These constructs can be thought of as the “mouths” of a wormhole, and so represent a instantaneous physical connection (“portals”) between the past and the future such that any quantum state which propagates into the future mouth immediately emerges from the corresponding past mouth. Note however that of course not every chronology-violating system is associated with a *wormhole-based* time machine, and so these connections should be taken as representing closed timelike curves in the most general, abstract sense.

5.2 List of special gates

Note that in these examples, d is the dimensionality of the relevant linear operator's underlying Hilbert space.

- X (Pauli-X):

$$\begin{array}{c} \text{---} \\ | \end{array} \boxed{\hat{X}} \begin{array}{c} \text{---} \\ | \end{array} \iff \hat{\sigma}_x = |0\rangle\langle 1| + |1\rangle\langle 0|$$

- Y (Pauli-Y):

$$\begin{array}{c} \text{---} \\ | \end{array} \boxed{\hat{Y}} \begin{array}{c} \text{---} \\ | \end{array} \iff \hat{\sigma}_y = -i|0\rangle\langle 1| + i|1\rangle\langle 0|$$

- Z (Pauli-Z):

$$\begin{array}{c} \text{---} \\ | \end{array} \boxed{\hat{Z}} \begin{array}{c} \text{---} \\ | \end{array} \iff \hat{\sigma}_z = |0\rangle\langle 0| - |1\rangle\langle 1|$$

- NOT (bit-flip), equivalent to the Pauli-X gate for qubits:

$$\begin{array}{c} \text{---} \\ | \end{array} \oplus \begin{array}{c} \text{---} \\ | \end{array} \iff \hat{\sigma}_x$$

- SUM (summation), a parameter-dependent d -dimensional generalization of the NOT gate:

$$\begin{array}{c} \text{---} \\ | \end{array} \boxed{\hat{\Sigma}(n)} \begin{array}{c} \text{---} \\ | \end{array} \iff \hat{\Sigma}(n) = \sum_{k=0}^{d-1} |k \oplus n\rangle\langle k|$$

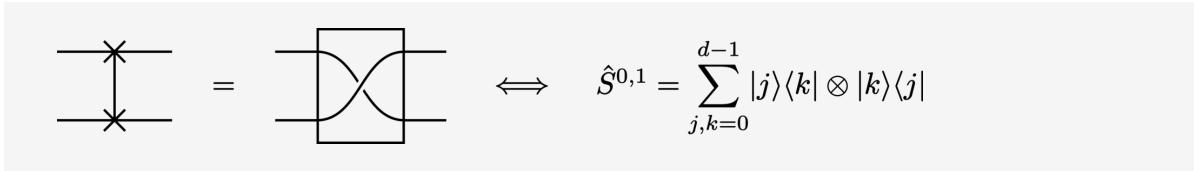
- P (phase):

$$\begin{array}{c} \text{---} \\ | \end{array} \boxed{\hat{P}(\omega)} \begin{array}{c} \text{---} \\ | \end{array} \iff \hat{P}(\omega) = \sum_{k=0}^{d-1} \omega^k |k\rangle\langle k|, \quad \omega \in \mathbb{C}, |\omega| = 1$$

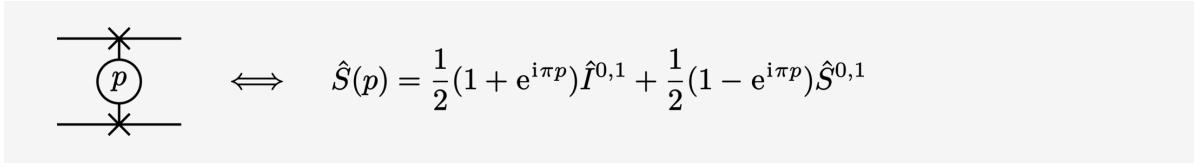
- R (rotation):

$$\begin{array}{c} \text{---} \\ | \end{array} \boxed{\hat{R}_k(\theta)} \begin{array}{c} \text{---} \\ | \end{array} \iff \begin{aligned} \hat{R}_x(\theta) &= \cos(\theta/2)(|0\rangle\langle 0| + |1\rangle\langle 1|) - i \sin(\theta/2)(|1\rangle\langle 0| + |0\rangle\langle 1|) &= e^{-i\hat{\sigma}_x\theta/2} \\ \hat{R}_y(\theta) &= \cos(\theta/2)(|0\rangle\langle 0| + |1\rangle\langle 1|) + \sin(\theta/2)(|1\rangle\langle 0| - |0\rangle\langle 1|) &= e^{-i\hat{\sigma}_y\theta/2} \\ \hat{R}_z(\theta) &= e^{-i\theta/2}|0\rangle\langle 0| + e^{i\theta/2}|1\rangle\langle 1| &= e^{-i\hat{\sigma}_z\theta/2} \end{aligned}$$

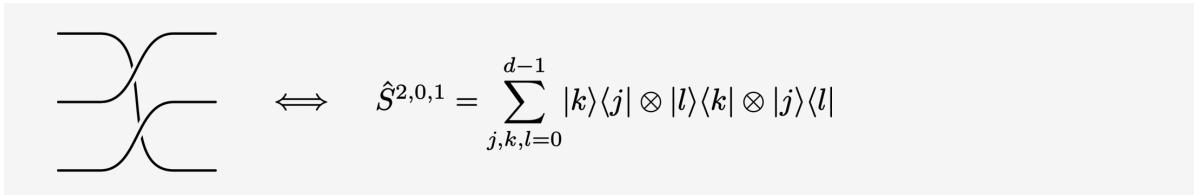
- SWAP (exchange):



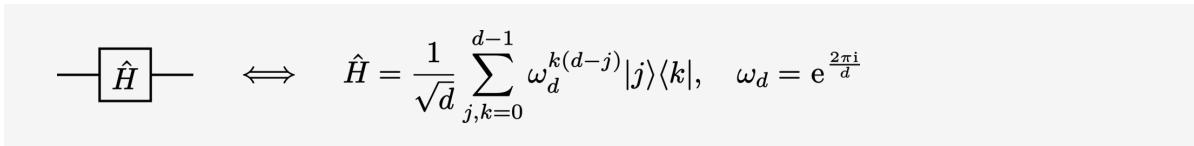
► PSWAP (power-SWAP):



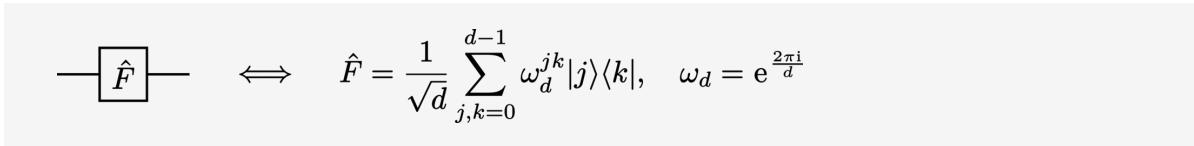
► PERM (permutation):



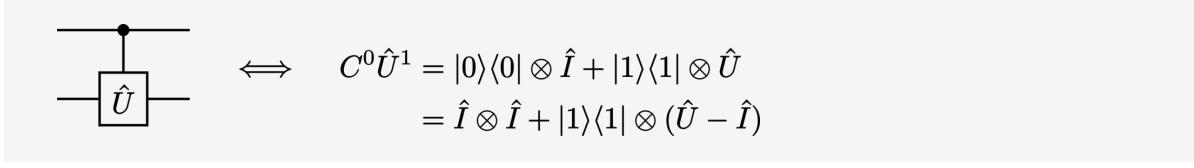
► HAD (Hadamard):



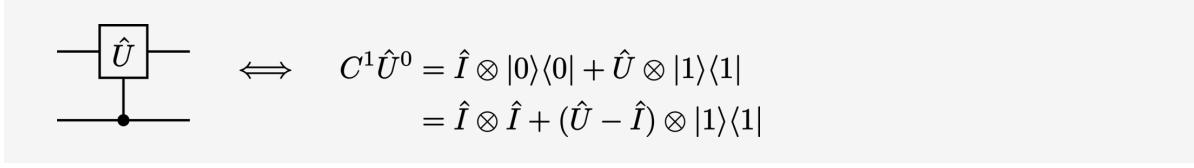
► QFT (quantum Fourier transform):



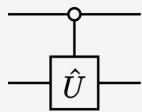
► controlled- \hat{U} :



► controlled- \hat{U} (inverted):

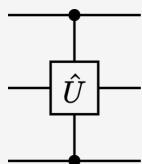


- anticontrolled- \hat{U} :



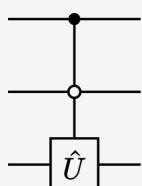
$$\iff \bar{C}^0 \hat{U}^1 = |1\rangle\langle 1| \otimes \hat{I} + |0\rangle\langle 0| \otimes \hat{U} \\ = \hat{I} \otimes \hat{I} + |0\rangle\langle 0| \otimes (\hat{U} - \hat{I})$$

- multiply-controlled- \hat{U} :



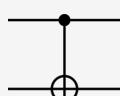
$$\iff C^{0,2} \hat{U}^1 = |0\rangle\langle 0|^0 \otimes \hat{I}^{1,2} + |1\rangle\langle 1|^0 \otimes C^2 \hat{U}^1 \\ = \hat{I}^{0,1} \otimes |0\rangle\langle 0|^2 + C^0 \hat{U}^1 \otimes |1\rangle\langle 1|^2 \\ = \hat{I} \otimes \hat{I} \otimes \hat{I} + |1\rangle\langle 1| \otimes (\hat{U} - \hat{I}) \otimes |1\rangle\langle 1|$$

- controlled-anticontrolled- \hat{U} :



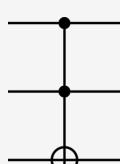
$$\iff C^0 \bar{C}^1 \hat{U}^2 = |0\rangle\langle 0|^0 \otimes \hat{I}^{1,2} + |1\rangle\langle 1|^0 \otimes \bar{C}^1 \hat{U}^2 \\ = \hat{I} \otimes \hat{I} \otimes \hat{I} + |1\rangle\langle 1| \otimes |0\rangle\langle 0| \otimes (\hat{U} - \hat{I})$$

- CNOT (controlled-NOT):



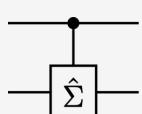
$$\iff C^0 \hat{X}^1 = |0\rangle\langle 0| \otimes \hat{I} + |1\rangle\langle 1| \otimes \hat{\sigma}_x$$

- CCNOT (controlled-controlled-NOT gate, doubly-controlled-NOT gate, also known as the *Toffoli* gate):



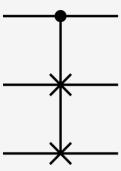
$$\iff C^{0,1} \hat{X}^2 = |0\rangle\langle 0|^0 \otimes \hat{I}^{1,2} + |1\rangle\langle 1|^0 \otimes C^1 \hat{X}^2 \\ = \hat{I} \otimes \hat{I} \otimes \hat{I} + |1\rangle\langle 1| \otimes |1\rangle\langle 1| \otimes (\hat{\sigma}_x - \hat{I})$$

- CSUM (controlled-SUM):



$$\iff C^0 \hat{\Sigma}^1 = \sum_{n,k=0}^{d-1} |n\rangle\langle n|^0 \otimes |k \oplus n\rangle\langle k|^1$$

- CSWAP (controlled-SWAP, also known as the *Fredkin* gate):



$$\iff C^0 \hat{S}^{1,2} = |0\rangle\langle 0|^0 \otimes \hat{I}^{1,2} + |1\rangle\langle 1|^0 \otimes \hat{S}^{1,2}$$

5.3 Examples

A better understanding of quantum circuitry notation can be gained by studying a few basic examples. Perhaps the simplest complete quantum circuit that we can construct consists of just a unipartite quantum state $\hat{\rho}$ which evolves into the future along a single quantum wire, subsequently becoming the output state $\hat{\rho}'$:

$$\hat{\rho} \xrightarrow{\quad} \hat{\rho}' \iff \hat{\rho}' = \hat{\rho}$$

As there are no operations (such as gates or measurements) on the wire, the explicitly labelled output state $\hat{\rho}'$ is identically equal to the input state $\hat{\rho}$ (since the quantum wire directly connects the two).

If we were to place a gate corresponding to an arbitrary unitary operator \hat{U} upon the intermediary wire, the action from the resulting operation is the standard transformation of a matrix by a linear operator:

$$\hat{\rho} \xrightarrow{\hat{U}} \hat{\rho}' \iff \hat{\rho}' = \hat{U}\hat{\rho}\hat{U}^\dagger$$

In the case of a pure (vector) input state, the circuit is simply:

$$|\phi\rangle \xrightarrow{\hat{U}} \hat{\rho}' \iff \hat{\rho}' = \hat{U}|\phi\rangle\langle\phi|\hat{U}^\dagger$$

In this case, we could, without loss of generality, denote the output state with a vector state (such as $|\phi'\rangle = \hat{U}|\phi\rangle$, equivalent to $\hat{\rho}' = |\phi'\rangle\langle\phi'|$), since the action of the (linear transformation) unitary gate preserves the purity of the systems on which it acts. Note that this is not true in general for all circuits; (non-unitary) operations like the (partial) trace do not necessarily preserve the purity of a particular system, and so the output states cannot always be denoted by vector states even if the input states are pure.

More interesting cases are those of multipartite systems. For instance, the evolution of a bipartite input state under a composite unitary gate is:

$$\hat{\rho} \left\{ \begin{array}{c} \xrightarrow{\hat{U}} \\ \hline \end{array} \right\} \hat{\rho}' \iff \hat{\rho}' = \hat{U}\hat{\rho}\hat{U}^\dagger$$

The evolution of the same state under parallel operations appears as:

$$\hat{\rho} \left\{ \begin{array}{c} \hat{U}_A \\ \hat{U}_B \end{array} \right\} \hat{\rho}' \iff \hat{\rho}' = (\hat{U}_A \otimes \hat{U}_B) \hat{\rho} (\hat{U}_A \otimes \hat{U}_B)^\dagger$$

In the case where the input state is explicitly separable, we write:

$$\hat{\rho}_A \quad \hat{\rho}_B \left\{ \begin{array}{c} \hat{U} \end{array} \right\} \hat{\rho}' \iff \hat{\rho}' = \hat{U} (\hat{\rho}_A \otimes \hat{\rho}_B) \hat{U}^\dagger$$

Due to the input state's separability, the mathematics corresponding to parallel operations on these systems may be greatly simplified:

$$\hat{\rho}_A \quad \hat{\rho}_B \left\{ \begin{array}{c} \hat{U}_A \\ \hat{U}_B \end{array} \right\} \hat{\rho}' \iff \hat{\rho}' = (\hat{U}_A \otimes \hat{U}_B) (\hat{\rho}_A \otimes \hat{\rho}_B) (\hat{U}_A \otimes \hat{U}_B)^\dagger = \hat{U}_A \hat{\rho}_A \hat{U}_A^\dagger \otimes \hat{U}_B \hat{\rho}_B \hat{U}_B^\dagger$$

This is, of course, simply equivalent to two distinct subcircuits with the outputs $\hat{\rho}'_A = \hat{U}_A \hat{\rho}_A \hat{U}_A^\dagger$ and $\hat{\rho}'_B = \hat{U}_B \hat{\rho}_B \hat{U}_B^\dagger$, respectively.

Note:

For more examples, see the *Examples* (page 199) section.

5.4 Remarks

5.4.1 Matrix vs. vector circuits

Note that our description of quantum circuits thus far has considered quantum processes involving both matrix states (such as mixed states and matrix representations of pure states) and vector states. Traditionally however, quantum circuit diagrams were used exclusively to visualize completely vector processes: all inputs are vector states, all intermediate operations are linear transformations, and so accordingly all outputs are vectors (up until any non-linear operations such as measurement). In such a paradigm, the mathematics corresponding to any given quantum circuit is necessarily always expressible as (linear and unitary) operations on *vectors*. It is therefore not necessary to express the circuit output using density matrices (such as outer products like $|\phi'\rangle\langle\phi'|$) in the case of exclusively purity-preserving operations. For example, we may write

$$|\phi\rangle \longrightarrow \boxed{\hat{U}} \longrightarrow |\phi'\rangle \iff |\phi'\rangle = \hat{U}|\phi\rangle$$

which is purely a vector description. In our combined matrix and vector formalism, we can equivalently express the above circuit as

$$|\phi\rangle \xrightarrow{\hat{U}} \hat{\rho}' \iff \hat{\rho}' = |\phi'\rangle\langle\phi'| = \hat{U}|\phi\rangle\langle\phi|\hat{U}^\dagger$$

where, despite the vector input and linear transformation, the output is given as a (density) matrix. The circuit formalism presented here is thus a more general one, such that the quantum circuits describe operations on (density) *matrix* states, not just vector states. This additionally means that both purity-preserving and purity-non-preserving operations (such as the partial trace) are completely captured by our methodology. Note however that in cases where a pure state output is obtained, it is often useful to specify such output in vector form (this being the most compact way of expressing pure states).

Albert Einstein's general theory of relativity is a geometric theory of gravitation which seeks to describe the effective mutual attraction between entities that possess mass. Within this theory, there exists an interesting class of objects called *closed timelike curves* (CTCs). Such objects are fascinating because a particle moving along one may travel back in time and interact with its past self, and so the geometries that permit the formation of CTCs are often colloquially known as *time machines*. It is because of this self-interaction that fictional depictions of time travel have captivated the interest of the public for more than a century, while both the validity and implications of the phenomenon have continually been discussed in the scientific literature.

6.1 Closed timelike curves and time machines

Closed timelike curves (CTCs) form an interesting class of objects within the general theory of relativity as they present the possibility for an observer to travel back in time. Formally, a CTC is an entirely geometric entity, being simply a curve (a path in spacetime) that is both closed and everywhere (either future- or past-directed) timelike. As such, in a universe with CTCs, any physical system may be able to interact with its past self, and so the potential existence of CTCs within our own universe naturally evokes scientific investigation into time travel. The foremost focus of such research is on questions regarding the non-trivial causal structure of CTC spacetimes and the consistency of the standard laws of physics.

In some locally unobjectionable exact solutions to the Einstein field equations, CTCs naturally occur [36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57]. However, many such spacetimes are often considered to be globally unphysical (like the classic swirling dust solutions of van Stockum [38] and Gödel [40]), and so the presence of interior CTCs is usually considered to be an artefact of the theory. Despite this, it is believed that CTCs may be constructed (at least theoretically) through the use of a traversable *wormhole* [58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83]. It is important to stress however that the physicality of such models is often called into question in the literature. This is due mainly to the fact that these solutions to the Einstein field equations often appear to necessitate the existence of matter possessing a negative energy density, and as such they do not always correspond to stable spacetimes. Note therefore that other solutions (such as the Kerr metric [43]), being free of the requirement of negative energy density, may enable the manifestation of CTCs in a much less contentious manner.

6.1.1 Wormhole-based time machines

In popular culture, wormholes (either artificially constructed or naturally occurring) commonly provide a means of interstellar (or even intergalactic) transportation. They accomplish this by allowing spaceships to quickly traverse their short, tubiform interiors and emerge at distant locations far sooner than any alternative non-wormhole routes would allow. Alternatively, if two separate moments in time were to be linked by wormhole instead of two separate locations, one would possess a *time machine* [61, 76, 84, 85, 86], i.e., a mechanism which can produce CTCs. Thus, wormholes are theoretically able to facilitate both long-distance space travel and time travel. It is this basis on which many of the studies involving wormhole-based time machines is established.

In essence, a wormhole is an exotic, theoretical object that connects two distant regions in space. The simplest version of a topological wormhole [87, 88, 89, 90] can be constructed by “cutting out” two spatially separated balls (denoted by \mathcal{W}^\pm , representing the two ends of the wormhole) in flat three-dimensional space and subsequently “gluing” (connecting) the surfaces (denoted by $\partial\mathcal{W}^\pm$) of these holes together. The resulting *mouths* (entrances) into the wormhole are thus *identified*, which is to say that entering into one results in simultaneous emergence from the other. See Figure 6.1 for a visual depiction of such geometry. Note that the resulting spacetime is topologically and geometrically non-trivial since

the space is not simply connected and the identified surfaces are not flat.

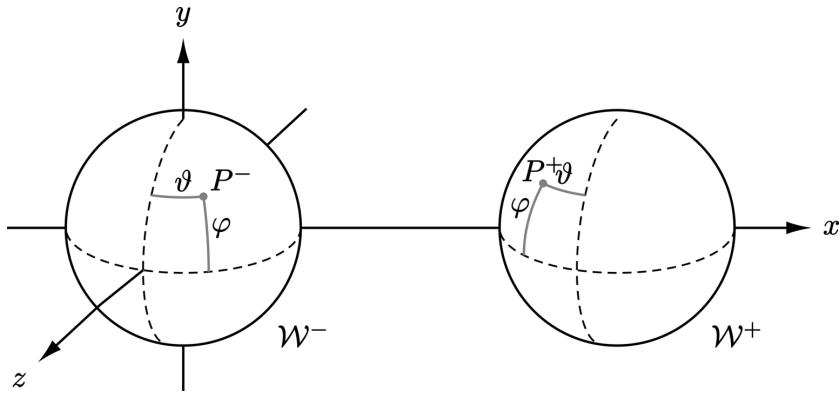


Figure 6.1: The surfaces of a simple topological wormhole.

To turn a wormhole into a time machine (thereby manifesting CTCs), a time “delay” has to be induced between the mouths. This can be accomplished in a few different ways, resulting in either of two distinct classes of time machine. The first class consists of those of which the time machine is *eternal* (meaning that CTCs are ever-present), and is accomplished simply by imposing a identification time difference (denoted Δt) between the two mouths (so that the coordinate time t on the “future” mouth \mathcal{W}^+ corresponds to the time $(t - \Delta t)$ on the “past” mouth \mathcal{W}^-). The second class consists of those created (theoretically) in a more physically realistic manner, such as moving one mouth (the future mouth) either into a strong gravitational field or accelerating it to relativistic speeds (and decelerating it back) [87]. For both classes of wormhole-based time machines, when the temporal difference between the mouths reaches a greater magnitude than their spatial separation, the mouths become causally connected, that is, CTCs manifest.

6.2 Time-travel paradoxes

The emergence of inconsistencies due to retro-causal action in the evolution of a time-travelling physical system is an issue which is captured by the infamous *grandfather paradox* (detailed in [66]). This problem, which is perhaps the foremost example of a time-travel paradox, takes on many forms, all of which share the common characteristic of antichronological causation. The archetype of the paradox often features an observer who travels to the past and, through their actions (usually by interfering with the relationship between their young grandparents), prevents their own birth. Consequently, the observer is unable to travel back in time to preclude their existence, thus the paradox is evident.

6.2.1 The principle of self-consistency

Time-travel paradoxes and the concept of retro-causality lie at the foundation of the issues with CTCs. In particular, the inconsistencies associated with paradoxical causal sequences have led to uncertainty regarding the pathology of CTCs and hence their research suitability. However, there is a distinct lack of evidence suggesting that the inconsistencies of CTC paradoxes are entirely inescapable. One may simply regard these problems as ill-posed, given that the initial conditions (e.g., the observer’s very existence and intent to intervene) are influenced by the future occurrences (e.g., observer stopping their grandparents from meeting). To resolve this problem, we could suppose that there exists some fundamental characteristic of reality which forbids any future event from paradoxically altering the past.

In particular, we could conjecture the existence of an innate law by which the universe operates, called the *principle of self-consistency* [87], which prohibits paradoxical causal sequences from transpiring. Under this condition, a globally consistent solution of the local physical laws must exist. This means that, while a system is allowed to propagate into its own past, it must do so in a way that is consistent with its own original history. Any interaction that occurs must be compatible with the past, and causal sequences containing events solely of this nature are characterized as being self-consistent.

6.3 The Cauchy problem on spacetimes containing CTCs

One way to explore the compatibility between the semiclassical laws of physics and CTCs is to determine whether the two can be consolidated without having unacceptable causality violations manifest. A prominent method with which this can be accomplished is by looking at the *Cauchy problem* [91, 92] for relativistic classical scalar fields in spacetimes containing CTCs. In this problem, one seeks to, among other things, prove or disprove the existence of the field solutions to the relevant time-evolution equations of motion given initial data. By determining the nature of these solutions, we are able to formulate conclusive remarks regarding the affinity of the union of semiclassical field theory and CTCs.

Interestingly, research into the Cauchy problem in spacetimes with CTCs has yielded some perhaps unanticipated results given the seemingly restrictive requirement of self-consistency. Foremost are the findings [87, 88, 89, 90, 93, 94, 95, 96, 97, 98], which show that globally consistent time evolutions for free relativistic classical scalar fields with generic initial data (posed before any regions with CTCs) exist and are generally unique in certain classes of non-globally hyperbolic (including CTC) spacetimes. Consequently, at least in the context of non-interacting, classical scalar fields, CTCs appear to be robust and spacetimes containing them are not as pathological as one might have initially suspected.

6.4 Time-travelling billiard balls

For a spacetime with one or more chronology-violating sets (regions containing CTCs), the past can be influenced by the future, provided that any changes which are made obey the self-consistency principle. The lack of such changes (e.g., self-interactions) in the Cauchy problem for non-interacting classical fields seems like a probable cause of its well-posedness on CTC spacetimes. Alternatively, a system that is able to self-interact (e.g., collide with its past/future self) is more likely to display unusual results—indeed, with an interacting field, uniqueness is thought to be lost as the strength of the interaction increases [90]. Therefore, in order to study this, models of interacting systems near CTCs were developed, and perhaps the most famous of these considers the elastic self-collision(s) of a time-travelling billiard ball. In this framework, a solid, elastic, spherical mass (the *billiard ball*) enters one mouth of a wormhole-based time machine, exits the other at an earlier time (due to a time-shift having been induced between the wormhole’s two mouths), and then collides with its earlier (past) self. Depending on the initial position and velocity of the ball, drastically distinct evolutions of the ball through the CTC-wormhole region can arise.

6.4.1 Indeterministic dynamics and evolution multiplicity

Studies [86, 87, 99, 100, 101, 102, 103] involving time-travelling billiard balls have shown that, given the same initial data posed in the presence of CTCs, there can be multiple self-consistent solutions which satisfy the equations of motion. Figure 6.2 illustrates a prominent example in which there are (at least) two distinct histories through which the billiard ball may evolve in a CTC-wormhole spacetime. This scenario is the foremost example of the *billiard-ball paradox*, and will hereafter be referred to as such. It is important to note that while the motivating formulation of this problem explicitly involves a wormhole-based time machine, the essence of the paradox is contingent only on the presence of CTCs that exactly facilitate the required function. In other words, while wormhole-based time machines make for perhaps the simplest conceptual model of CTCs, the physical mechanism behind the antichronological time travel in the billiard-ball paradox is irrelevant.

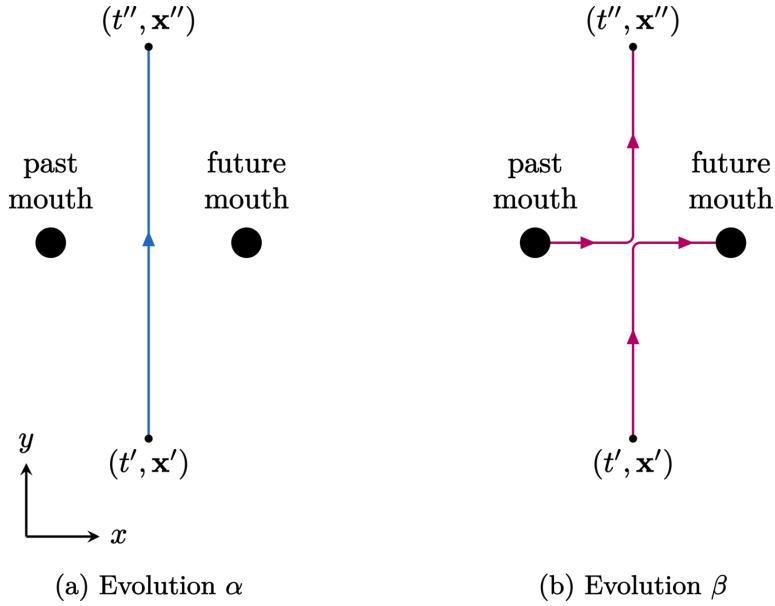


Figure 6.2: The billiard-ball paradox.

Unlike in the case of the grandfather paradox, the paradoxical issue here is not of self-inconsistent trajectories but is of the indeterminism in the self-consistent ones. This is to say that solutions always exist because they self-adjust themselves (thereby providing consistency), but we now have a different issue, that of solution multiplicity. This new problem is an interesting one, as the existence of more than one solution to the equations of motion contrasts with the determinism typically associated with classical mechanics.

6.4.2 Quantum resolutions to classical indeterminism

It is natural to question whether the characteristic of solution multiplicity is indeed merely an artefact of a classical description and vanishes in a fully self-consistent quantum mechanical treatment, or whether it remains as a pathological aspect of CTCs and time travel under any physical description. In past work [87], it has been suggested that the indeterminism problem of the billiard-ball paradox disappears in quantum theory. The reasoning behind this is that, unlike classical mechanics, a quantum treatment provides one with ways of assigning solution probabilities, thereby replacing the classical theory's multiplicity of solutions with a set of probabilities for the outcomes of all sets of measurements.

For example, a quantum prescription based on the path-integral formulation tells us that classical trajectories coexist in a probabilistic sense. In a prominent paper by Friedman et al. [87], the authors discuss such a prescription applied to the exact evolutions depicted in Figure 6.2. When the ball begins in a nearly classical wave-packet, Friedman et al. postulate that a WKB (Wentzel-Kramers-Brioullin) approximation to their sum-over-histories method would find that the wave-packet emerges from the CTC-wormhole region having travelled along either evolution α or β with equal (i.e., 50%) probability. One interpretation of this is to say that the associated quantum billiard ball travels along *both* paths simultaneously through the time-machine region, thus ameliorating the indeterminism of classical mechanics. However, no calculation is provided in [87] or any subsequent papers to support the WKB approximation conjecture. Nonetheless, the pioneering semiclassical method proposed by Friedman et al. motivates further study of this problem, as the interesting probabilistic result they postulated is simply inaccessible to classical mechanics.

Quantum theories of time travel

The advent of CTCs and their associated issues in the semiclassical general theory of relativity prompted study into quantum theories of time travel. In the absence of a complete quantum theory of gravity, reconciling CTCs with standard quantum mechanics forms a compelling basis for research. Indeed, exploration into the interplay between quantum mechanics and CTCs, even in only a theoretical manner, may provide insight into a yet unknown full theory of quantum gravity. Thus far, research into this area has taken several main routes, all of which have produced incredibly fascinating results.

Perhaps the most immediately interesting research direction is the study of the various ways by which ordinary quantum mechanics can be extended to include antichronological time travel. Such theories, called *prescriptions*, are simply models of quantum state evolution on or near CTCs which employ self-consistency (often Novikov's *principle of self-consistency*) in order to compute the evolved states and their histories. For so-called "paradoxical" interactions—those for which the interaction between the incoming *chronology-respecting* (CR) state and its *chronology-violating* (CV) counterpart on the CTC seemingly describes a temporal paradox—solutions for these states that satisfy the relevant prescription's equations of motion for any given interaction are termed *resolutions*.

The two main such prescriptions of quantum time travel are the *Deutsch model* (giving *Deutschian* CTCs, or D-CTCs), in which self-consistency is applied to the density matrix itself, and *postselected teleportation* (giving P-CTCs), which is equivalent to a path-integral formulation. Note that alternative prescriptions of quantum time travel do exist, but none are as mature as either D-CTCs or P-CTCs, and so a select handful are discussed only briefly in *Alternative formulations of quantum time travel* (page 72). In any case, it is the exploration into these models and their consequences that is most pertinent to our concerns in this project, as this is perhaps the research direction in which the most conclusions can be reached in regards to the quantum information processing and computation abilities of CTCs.

Other research directions into the quantum mechanics of time travel take distinctly different approaches. These include various efforts to formulate a self-consistent version of quantum theory near or on CTCs [104, 105, 106, 107, 108], generalize relativistic quantum field theory to non-globally hyperbolic spacetimes (i.e., those containing CTCs) [109, 110, 111, 112, 113, 114], pose and solve the Cauchy problem for quantum fields on such spacetimes [115], and experimentally simulate the various quantum models of closed timelike curves [116, 117, 118]. In particular, studies of *interacting* quantum systems near CTCs have concluded that, even on non-globally hyperbolic spacetimes, the interactions of quantum particles interpolate (depending on the interaction strength) between plane-wave scattering events (for weak interaction strengths) and hard-sphere collisions (for strong interaction strengths), exactly as one might expect for ordinary globally hyperbolic spacetimes. However, for large classes of fields and states, the corresponding quantum evolutions fail to be unitary [104, 105, 106, 107, 115, 119, 120, 121, 122, 123], which is especially true in the case of strongly interacting fields. This jeopardizes the notion of obtaining a consistent probability interpretation of quantum mechanics—a serious consequence which arises specifically as ambiguities in the assignment of probabilities for events occurring *before* (or even spacelike-separated from) the region with CTCs.

On the other hand, recent studies of expressly unitary (albeit non-interacting) quantum mechanics on CTCs [124, 125] have yielded results which suggest that thermal fluctuations destroy causation and erase all memories of systems evolving along CTCs, even those of macroscopic observers. As a consequence, the resulting quantum theory is perfectly self-consistent, in the sense that all states undergo non-contradictory histories and all retro-causality paradoxes (i.e., time-travel paradoxes) are forbidden at the quantum level.

Less direct (yet still important) approaches of studying (antichronological) time travel and causality involve exploring various associated or complimentary aspects of these notions in both quantum and relativistic capacities. Such topics include:

- ▶ CTC spacetimes and their stability [36, 39, 40, 41, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 54, 56, 57, 126, 127, 128, 129, 130, 131, 132, 133, 134]

- ▶ chronology protection and topological censorship [135, 136, 137, 138, 139, 140]
- ▶ (non-)global hyperbolicity [90, 93, 94, 95, 96, 98, 109, 110, 111, 113, 114, 141]
- ▶ thermodynamics [124, 142, 143]
- ▶ causality [46, 107, 130, 131, 132, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175]
- ▶ energy conditions [61, 71, 72, 129, 144, 176, 177, 178, 179, 180]
- ▶ wormhole traversability [55, 62, 65, 67, 69, 70, 71, 72, 74, 75, 77, 78, 80, 82, 83, 181]
- ▶ superluminality [128, 176, 177, 179, 182]
- ▶ locality [109, 111]
- ▶ holography [183]

Note however that these topics are mentioned purely for the sake of completeness. Since we are primarily interested in computing state evolutions using the aforementioned quantum prescriptions of time travel, all of the concerns in these topics are out of the scope of this project.

7.1 The Deutsch model (D-CTCs)

In 1991, David Deutsch introduced a method of analysing the flow of information in a chronology-violating network. This prescription, now known as the *Deutsch model* [184], consists of a quantum theory of computation applied to a CTC model where most of the geometry has been abstracted away. It is based on the requirement that time evolution of the local density operator (taken as a fundamental object) of a quantum system must be self-consistent.

One of the main results of Deutsch's work on time travel is his proposal for the time evolution of quantum states through a quantum circuit representation of a chronology-violating network (illustrated in Figure 7.1), with particular note as to how it resolves time-travel paradoxes and indeterminism.

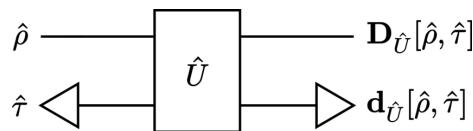


Figure 7.1: Deutsch's model of quantum time travel.

The mathematical formulation of his self-consistent evolution of a quantum state through a region with a time machine is provided by requiring that any given chronology-violating (CV) system enters the CTC in the state

$$\hat{\tau} \in \mathcal{D}(\mathcal{H}_{\text{CV}}), \quad (7.1)$$

and then emerges in the past in the same state despite having interacted with a chronology-respecting (CR) system in the state

$$\hat{\rho} \in \mathcal{D}(\mathcal{H}_{\text{CR}}) \quad (7.2)$$

through a unitary \hat{U} on $\mathcal{H}_{\text{CR}} \otimes \mathcal{H}_{\text{CV}}$. The joint density operator entering the gate is simply the tensor product $\hat{\rho} \otimes \hat{\tau}$ on $\mathcal{H}_{\text{CR}} \otimes \mathcal{H}_{\text{CV}}$ so that the standard evolution through the unitary is given by

$$\hat{\rho} \otimes \hat{\tau} \rightarrow \mathbf{E}_{\hat{U}}[\hat{\rho} \otimes \hat{\tau}] = \hat{U}(\hat{\rho} \otimes \hat{\tau})\hat{U}^\dagger. \quad (7.3)$$

We can now trace the CV and CR subsystems out and denote the resulting maps as

$$\mathbf{d}_{\hat{U}}[\hat{\rho}, \hat{\tau}] \equiv \text{tr}_{\text{CR}}[\hat{U}(\hat{\rho} \otimes \hat{\tau})\hat{U}^\dagger], \quad (7.4)$$

$$\mathbf{D}_{\hat{U}}[\hat{\rho}, \hat{\tau}] \equiv \text{tr}_{\text{CV}}[\hat{U}(\hat{\rho} \otimes \hat{\tau})\hat{U}^\dagger]. \quad (7.5)$$

Self-consistent solutions of the CTC state may then be identified as fixed points of the CV map (7.4), which are expressible via

$$\hat{\tau} = \mathbf{d}_{\hat{U}}[\hat{\rho}, \hat{\tau}]. \quad (7.6)$$

This condition essentially codifies the requirement that the “younger” CTC state (the right-hand side) exiting the gate is the same as the “older” CTC state (the left-hand side) entering the gate. Once solutions to equation (7.6) are determined, the evolution of the system state $\hat{\rho}$ through the interaction \hat{U} in this Deutsch model is then simply given by the CR map (7.5). Deutsch’s original construction specifically interprets D-CTCs as being portals between parallel universes—a time traveller on a D-CTC would pass in and out of such universes—and so the quantum states in this context describes a superposition of states where the time traveller does and does not exist.

The fact that the imposed self-consistency constraint in (7.6) is a fixed-point condition means that the non-linearity introduced to the laws of quantum mechanics is distinct from that of other models. Furthermore, as the rigid matching requirement is for the density matrix as opposed to the individual states in a pure state decomposition, the Deutsch model intrinsically treats density matrices ontologically as ontic (“real”) objects rather than as epistemic knowledge representations.

7.1.1 Deutsch’s picture: The rule of maximum entropy

Of crucial significance is the fact that the Deutsch prescription resolves time-travel paradoxes without placing any constraints on the input system state. This is because Deutsch proved that at least one consistent solution of the (normalized) map

$$\overline{\mathbf{d}_{\hat{U}}}[\hat{\rho}, \hat{\tau}^{(0)}] \equiv \lim_{N \rightarrow \infty} \frac{1}{N+1} \sum_{k=0}^N \mathbf{d}_{\hat{U}}^k(\hat{\rho}, \hat{\tau}^{(0)}) \quad (7.7)$$

exists for every unitary \hat{U} and input $\hat{\rho}$. Here, $\hat{\tau}^{(0)} \in \mathcal{L}(\mathcal{H}_{\text{CV}})$ represents some initial CV state, and \mathbf{d}^k denotes k compositions of the D-CTC CV map, i.e.,

$$\begin{aligned} \mathbf{d}_{\hat{U}}^k[\hat{\rho}, \hat{\tau}^{(0)}] &= (\underbrace{\mathbf{d}_{\hat{U}} \circ \mathbf{d}_{\hat{U}} \circ \dots \circ \mathbf{d}_{\hat{U}}}_{k \text{ times}})[\hat{\rho}, \hat{\tau}^{(0)}] \\ &= \mathbf{d}_{\hat{U}} \left[\hat{\rho}, \mathbf{d}_{\hat{U}} \left[\hat{\rho}, \dots, \mathbf{d}_{\hat{U}} \left[\hat{\rho}, \hat{\tau}^{(0)} \right] \right] \right]. \end{aligned} \quad (7.8)$$

Note however that some interactions have a multiplicity of fixed points, which leads to non-determinism in the time evolution. This characteristic, known as the *uniqueness ambiguity* in the context of D-CTCs, may be interpreted as an indication that the theory is incomplete.

To overcome this problem, Deutsch argued that the “correct” solution is the one with the most von Neumann entropy (3.43). This proposition appeals to the principles of thermodynamics (the second law in particular), and asserts that the trapped CTC would naturally tend towards a maximally entropic equilibrium state. Given its lack of rigorous theoretical support however, Deutsch’s rule of maximum entropy is often considered to be an unsatisfactory approach to overcoming the problem of plurality in the CTC state solution.

7.1.2 Iterative picture: The equivalent circuit

In 2010, Ralph and Myers introduced their equivalent-circuit picture (ECP) [76, 185, 186, 187, 188] of the D-CTC model. This approach was formulated as a method of deriving the maximum entropy rule and thus resolving the uniqueness ambiguity. It consists of essentially “unwrapping” the D-CTC time-loop fixed-point circuit (Figure 7.1) into an infinite sequence of interactions with perfect copies of the input state, and is illustrated in Figure 7.2.

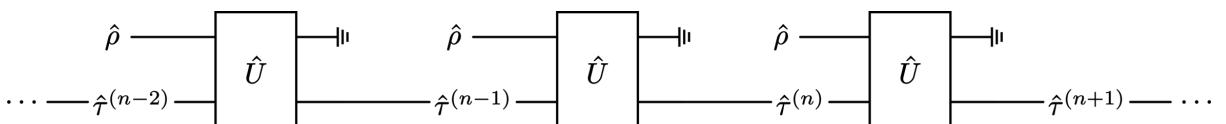


Figure 7.2: The equivalent-circuit picture of a D-CTC.

In this model, the CTC-trapped state $\hat{\tau}^{(n-1)}$ enters the n th interaction \hat{U} and exits, according to the fixed-point condition (7.6), in the state

$$\hat{\tau}^{(n)} = \mathbf{d}_{\hat{U}}[\hat{\rho}, \hat{\tau}^{(n-1)}]. \quad (7.9)$$

The final output $\hat{\tau}^{(\infty)}$ is then obtained by applying this map recursively *ad infinitum* to an initial state $\hat{\tau}^{(0)}$. Accordingly, one finds, in the limit,

$$\hat{\tau}^{(\infty)} = \lim_{N \rightarrow \infty} \mathbf{d}_{\hat{U}}^N[\hat{\rho}, \hat{\tau}^{(0)}] \quad (7.10)$$

Given this framework, the ECP then introduces an arbitrarily small decoherence interaction \mathcal{D} , parametrized by the decoherence strength $0 < \delta \ll 1$, to each CTC iteration,

$$\begin{aligned} \hat{\tau}^{(n)} &= \mathcal{D}[\mathbf{d}_{\hat{U}}[\hat{\rho}, \hat{\tau}^{(n-1)}]] \\ &= (1 - \delta)\mathbf{d}_{\hat{U}}[\hat{\rho}, \hat{\tau}^{(n-1)}] + \delta\bar{\hat{I}} \end{aligned} \quad (7.11)$$

where

$$\begin{aligned} \bar{\hat{I}} &\equiv \frac{1}{\dim(\mathcal{H})}\hat{I}, \\ \hat{I} &\equiv \sum_i^{\dim(\mathcal{H})} |i\rangle\langle i| \in \mathcal{H}, \end{aligned} \quad (7.12)$$

are the unnormalized and normalized maximally mixed (identity) states, respectively. The corresponding final output is then

$$\hat{\tau}^{(\infty)} = \lim_{N \rightarrow \infty} \left[(1 - \delta)^N \mathbf{d}_{\hat{U}}^N[\hat{\rho}, \hat{\tau}^{(0)}] + \sum_{k=0}^{N-1} \delta^k (1 - \delta)^k \mathbf{d}_{\hat{U}}^k[\hat{\rho}, \bar{\hat{I}}] \right]. \quad (7.13)$$

From this result, it can be determined that the non-uniqueness in the solutions to the CV CTC map corresponds to sensitivity to the initial condition $\hat{\tau}^{(0)}$. Ralph and Myers then show that the result of (7.13) converges to the unique solution of Deutsch's maximum entropy rule, and that this rule and result coincides with setting the initial state to be the maximally mixed CTC state $\hat{\tau}^{(0)} = \bar{\hat{I}}$ in (7.10). The appealing feature of this picture is that Deutsch's maximum entropy rule is not merely a postulate of the theory; rather, it is a direct result.

Later, Allen [23] noticed that the equivalent circuit does not always converge when there is decoherence interaction, and that Ralph and Myers's proof of the derivation of the maximum entropy rule was incomplete. However, Dong [188] then showed that the ECP leads instead to a *revised* maximum entropy rule, one which is distinct from (and perhaps more valid than) Deutsch's original rule. It asserts that, when multiple valid solutions of the CV map exist, the Deutsch model "chooses" the *stable* fixed-point state with the most entropy (as determined by the equivalent model with a decoherence interaction). Additionally, note that the final fixed point as given by this revised rule is in general dependent on the decoherence strength (δ).

7.1.3 Allen's picture: Noise incorporation

In 2014, Allen [23] proposed a different method of resolving the uniqueness ambiguity. It consists of the incorporation of noise into the CV channel via a noise channel \mathcal{N} with depolarization parameter $0 < \nu < 1$, which is accomplished by

$$\begin{aligned} \hat{\tau} &= \mathcal{N}[\mathbf{d}_{\hat{U}}[\hat{\rho}, \hat{\tau}]] \\ &= (1 - \nu)\mathbf{d}_{\hat{U}}[\hat{\rho}, \hat{\tau}] + \nu\bar{\hat{I}}. \end{aligned} \quad (7.14)$$

Allen then proves that this addition resolves the uniqueness ambiguity without requiring a maximum entropy rule, and that the unique fixed-point CTC state is $\hat{\tau}^{(0)}$ -independent and given by

$$\overline{\mathcal{N}[\mathbf{d}_{\hat{U}}^\infty]}(\hat{\rho}) = \lim_{N \rightarrow \infty} \frac{1}{N+1} \sum_{k=0}^N \mathcal{N}[\mathbf{d}_{\hat{U}}^k[\hat{\rho}, \hat{\tau}^{(0)}]]. \quad (7.15)$$

Given that \mathcal{N} is a noise channel, then it should be *strictly contractive* (that is, its trace distance should always decrease under its action), leaving (7.15) unique. Dong [188] then show that Allen's picture is inequivalent to both Deutsch's and Ralph and Myers's pictures.

7.2 Postselected teleportation (P-CTCs)

The other main prescription of the quantum mechanics of time travel, *postselected teleportation* (P-CTCs) [116, 189] (based on unpublished work in [190], and also due independently to [191] as discussed in [192]), also provides self-consistent resolutions to general time-travel paradoxes. P-CTCs are motivated as mechanisms for facilitating the quantum teleportation of quantum systems to the past [190]. The contemporary mathematical description of the prescription is based upon postselection, and essentially consists of a quantum communication channel that is formed through teleportation.

The CV (CTC) quantum channel is replaced with a maximally entangled state $|\Phi\rangle$ (i.e., a Bell state for qubits), allowing for states to be transported between a sender and receiver through shared entanglement. Quantum teleportation further combined with postselection results in a quantum channel to the past. Antichronological time travel then manifests due to the entanglement between the forward and backward parts of the curve, and one thus obtains a possible theoretical description of quantum mechanics near CTCs. Note that the replacement of the CTC channel with the postselection of a maximally entangled state is equivalent to a path-integral formulation (developed in [106]) in which the CTC state's wave function evolves in accordance with self-consistent boundary conditions (specifically, the initial and final coordinates are the same). This is why we say that P-CTCs has the important feature of being equivalent to the path-integral formulation of quantum mechanics. Here, we demonstrate the model.

7.2.1 The path-integral formulation in the presence of CTCs

The path-integral formulation of quantum mechanics allows us to formulate the transition amplitude for a quantum system which evolves from an initial state $|\psi_i\rangle$ to a final state $|\psi_f\rangle$ as an integral over the paths of the action. In one spatial dimension $x(t)$, we may simply write

$$\langle\psi_f|\hat{U}(t'',t')|\psi_i\rangle = \iint dx' dx'' \psi_i(x') \psi_f^*(x'') \int_{x'}^{x''} \mathcal{D}'x e^{iS[x]/\hbar}. \quad (7.16)$$

Here, $\hat{U}(t'',t')$ is the time-evolution operator that describes the state evolution from initial time t' to final time t'' ,

$$\begin{aligned} x(t') &\equiv x', \\ x(t'') &\equiv x'', \end{aligned} \quad (7.17)$$

are the initial and final coordinates respectively,

$$\begin{aligned} |\psi_i\rangle &= \int dx' \psi_i(x') |x'\rangle, \\ |\psi_f\rangle &= \int dx'' \psi_f(x'') |x''\rangle, \end{aligned} \quad (7.18)$$

are the position representations of the initial and final states, and

$$S[x] \equiv \int_{t'}^{t''} L(t, x(t), \dot{x}(t)) dt, \quad (7.19)$$

is the action of path $x(t)$ with Lagrangian $L(t, x(t), \dot{x}(t))$.

To incorporate a CTC into the path-integral formulation, we first divide the spacetime into two parts, or more formally, two separate systems: a CR system $|\psi\rangle \in \mathcal{H}_{\text{CR}}$ and a CV system $|\phi\rangle \in \mathcal{H}_{\text{CV}}$. Let their position-space representations be parametrized by the coordinates u and v respectively, so that the initial and final position representations are

$$\begin{aligned} |\psi_i\rangle &= \int du' \psi_i(u') |u'\rangle & \rightarrow & \quad |\psi_f\rangle = \int du'' \psi_i(u'') |u''\rangle, \\ |\phi_i\rangle &= \int dv' \phi_i(v') |v'\rangle & \rightarrow & \quad |\phi_f\rangle = \int dv'' \phi_i(v'') |v''\rangle. \end{aligned} \quad (7.20)$$

This means that, with time-evolution unitary $\hat{U}(t', t'')$ on $\mathcal{H}_{\text{CR}} \otimes \mathcal{H}_{\text{CV}}$, the joint input state evolves as per $|\psi_i\rangle \otimes |\phi_i\rangle \rightarrow |\psi_f\rangle \otimes |\phi_f\rangle$ over the spatial coordinates $(u', v') \rightarrow (u'', v'')$. Using equation (7.16), this

evolution has a corresponding path-integral representation of

$$(\langle \psi_f | \otimes \langle \phi_f |) \hat{U}(t'', t') (|\psi_i\rangle \otimes |\phi_i\rangle) = \iiint du' du'' dv' dv'' \psi_i(u') \psi_f^*(u'') \phi_i(v') \phi_f^*(v'') \int_{(u', v')}^{(u'', v'')} \mathcal{D}'x e^{iS[x]/\hbar}. \quad (7.21)$$

Now, to include a CTC in this scheme, we mandate that the initial and final coordinates for the chronology-violating system are the same, just like in a CTC. Mathematically, we impose the boundary condition $v' = v''$ a δ -function in (7.21), and the resulting transition amplitude for the chronology-respecting system with an incorporated CTC may be written as

$$(\langle \psi_f | \otimes \hat{I}) \hat{U}(t'', t') (|\psi_i\rangle \otimes \hat{I}) = \iiint du' du'' dv' dv'' \psi_i(u') \psi_f^*(u'') \delta(v' - v'') \int_{(u', v')}^{(u'', v'')} \mathcal{D}'x e^{iS[x]/\hbar}. \quad (7.22)$$

The coherent addition of all possible initial and final conditions for the CTC is captured through the v' and v'' integrals, hence the removal of the CTC states $|\phi_i\rangle$ and $|\phi_f\rangle$. This indicates that it is not possible to assign a state to the CTC system (at least in this context), which in turn means that compatibility with the boundary conditions implies that the CR system is not in a forbidden state.

7.2.2 Equivalence of CTC path integrals and postselected teleportation

Given the form (7.22) of the transition amplitude obtained using path-integral methods, we now wish to show that it is equal to the form which one obtains from postselected teleportation methods. To do this, let us first define our maximally entangled bipartite state of choice $|\Phi\rangle \in \mathcal{H}_{\text{CV}} \otimes \mathcal{H}_{\text{T}}$ in a w -space representation as

$$|\Phi\rangle = \int dw \Phi(w) |w\rangle \otimes |w\rangle \quad (7.23)$$

where we assume that $\Phi(w)$ satisfies the normalization condition

$$1 = \langle \Phi | \Phi \rangle = \int dw |\Phi(w)|^2. \quad (7.24)$$

Furthermore, note how (7.23) can be put in a more useful form by using a δ -function,

$$|\Phi\rangle = \iint dv dw \Phi(v, w) \delta(v - w) |v\rangle \otimes |w\rangle. \quad (7.25)$$

From here, we replace the CV state $|\phi\rangle$ with the maximally entangled $|\Phi\rangle$, thereby meaning that our input state $|\psi_i\rangle \otimes |\phi_i\rangle$ evolves under a transformation described by the composition $\hat{U}(t'', t') \otimes \hat{I}$. To clarify, the time-evolution operator \hat{U} only acts on the \mathcal{H}_{CR} (CR) and \mathcal{H}_{CV} (CV) Hilbert spaces while the identity \hat{I} acts on the third (teleportation) Hilbert space \mathcal{H}_{T} . With this, we then postselect on the output being in the state $|\psi_f\rangle \otimes |\Phi\rangle$, which physically represents the teleportation Hilbert space \mathcal{H}_{T} being used to send this conditional selection back in time, thereby yielding the necessary postselected teleportation. This entire sequence can be formulated by replacing the CTC state in (7.21) with our

maximally entangled state (7.25), which yields

$$\begin{aligned}
 (\langle\psi_f| \otimes \langle\Phi|)(\hat{U}(t'', t') \otimes \hat{I})(|\psi_i\rangle \otimes |\Phi\rangle) &= \iint dv'' dw'' \Phi^*(v'', w'') \delta(v'' - w'') (\langle\psi_f| \otimes \langle v''| \otimes \langle w''|) \\
 &\quad \times (\hat{U}(t'', t') \otimes \hat{I}) \\
 &\quad \times \iint dv' dw' \Phi(v', w') \delta(v' - w') (|\psi_i\rangle \otimes |v'\rangle \otimes |w'\rangle) \\
 &= \iiint dv' dv'' dw' dw'' \Phi(v', w') \Phi^*(v'', w'') \delta(v' - w') \delta(v'' - w'') \\
 &\quad \times (\langle\psi_f| \otimes \langle v''| \otimes \langle w''|) (\hat{U}'(t'', t') \otimes \hat{I}) (|\psi_i\rangle \otimes |v'\rangle \otimes |w'\rangle) \\
 &= \iiint dv' dv'' dw' dw'' \Phi(v', w') \Phi^*(v'', w'') \delta(v' - w') \delta(v'' - w'') \\
 &\quad \times \iint du' du'' \psi_i(u') \psi_f^*(u'') \int_{(u', v')}^{(u'', v'')} \mathcal{D}'x e^{iS[x]/\hbar} \cdot \underbrace{\langle w''| \hat{I}| w'\rangle}_{=\delta(w'' - w')} \\
 &= \iiint dv' dv'' dw' \Phi(v', w') \Phi^*(v'', w') \delta(v' - w') \delta(v'' - w') \quad (7.26) \\
 &\quad \times \iint du' du'' \psi_i(u') \psi_f^*(u'') \int_{(u', v')}^{(u'', v'')} \mathcal{D}'x e^{iS[x]/\hbar} \\
 &= \iint dv' dv'' \Phi(v') \Phi^*(v'') \delta(v' - v'') \\
 &\quad \times \iint du' du'' \psi_i(u') \psi_f^*(u'') \int_{(u', v')}^{(u'', v'')} \mathcal{D}'x e^{iS[x]/\hbar} \\
 &= \iiint du' du'' dv' dv'' \psi_i(u') \psi_f^*(u'') \delta(v' - v'') \\
 &\quad \times \int_{(u', v')}^{(u'', v'')} \mathcal{D}'x e^{iS[x]/\hbar}.
 \end{aligned}$$

Thus, we can see that the path-integral formulation (7.22) is equivalent to postselection (7.26), that is,

$$(\langle\psi_f| \otimes \hat{I})\hat{U}(t'', t')(|\psi_i\rangle \otimes \hat{I}) = (\langle\psi_f| \otimes \langle\Phi|)(\hat{U}(t'', t') \otimes \hat{I})(|\psi_i\rangle \otimes |\Phi\rangle). \quad (7.27)$$

Of course, being based on the path-integral formulation, these quantities are in general not normalized, and so must be renormalized in order to ensure that the probabilistic interpretation of the theory remains intact.

7.2.3 General form of P-CTC system evolution

Here we show how a CR system state $\hat{\rho} \in \mathcal{H}_{\text{CR}}$ in the Dirac bra-ket formalism evolves through the postselected teleportation chronology-violating network depicted in Figure 7.3.

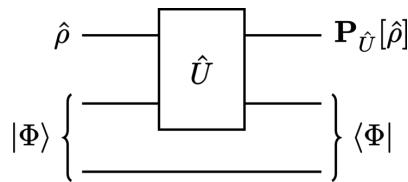


Figure 7.3: The postselected teleportation model of quantum time travel.

In the P-CTCs formalism, self-consistent resolutions to quantum time-travel paradoxes are provided by way of quantum teleportation. This is achieved by first introducing a teleportation mode to the pre-existing CR and CV pair. On the CV and teleportation modes, a maximally entangled state $|\Phi\rangle \in \mathcal{P}(\mathcal{H}_{\text{CV}} \otimes \mathcal{H}_{\text{T}})$ is placed (preselected). Here, \mathcal{H}_{T} is the Hilbert space of the teleportation subsystem. With this, the evolution of the product input state $\hat{\rho} \otimes |\Phi\rangle\langle\Phi|$ under the unitary $\hat{U} \otimes \hat{I}$ is simply

$$\hat{\rho} \otimes |\Phi\rangle\langle\Phi| \rightarrow \mathbf{E}_{\hat{U} \otimes \hat{I}}[\hat{\rho} \otimes |\Phi\rangle\langle\Phi|] = (\hat{U} \otimes \hat{I})(\hat{\rho} \otimes |\Phi\rangle\langle\Phi|)(\hat{U}^\dagger \otimes \hat{I}). \quad (7.28)$$

In accordance with the protocol of quantum teleportation, subsequent postselection against the same entangled state results in a quantum channel to the past (on the subsystem \mathcal{H}_T). Mathematically, the evolved CR state becomes

$$\begin{aligned}\mathbf{P}_{\hat{U}}[\hat{\rho}] &\propto (\hat{I} \otimes \langle \Phi |) (\hat{U} \otimes \hat{I}) (\hat{\rho} \otimes |\Phi\rangle\langle\Phi|) (\hat{U}^\dagger \otimes \hat{I}) (\hat{I} \otimes |\Phi\rangle) \\ &= \text{tr}_{\text{CV}}[\hat{U}] \hat{\rho} \text{tr}_{\text{CV}}[\hat{U}^\dagger],\end{aligned}\quad (7.29)$$

where we computed

$$(\hat{I} \otimes \langle \Phi |) (\hat{U} \otimes \hat{I}) (\hat{I} \otimes |\Phi\rangle) \propto \text{tr}_{\text{CV}}[\hat{U}]. \quad (7.30)$$

The P-CTCs evolution of the input state $\hat{\rho}$ is then given by the renormalized form of the non-unitary map (7.29), that is,

$$\mathbf{P}_{\hat{U}}[\hat{\rho}] = \frac{\hat{\mathcal{P}} \hat{\rho} \hat{\mathcal{P}}^\dagger}{\text{tr}[\hat{\mathcal{P}} \hat{\rho} \hat{\mathcal{P}}^\dagger]}. \quad (7.31)$$

where we defined the P-CTC operator

$$\hat{\mathcal{P}} \equiv \text{tr}_{\text{CV}}[\hat{U}]. \quad (7.32)$$

In the case of a pure CR input state $\hat{\rho} = |\psi\rangle\langle\psi|$, the associated P-CTC CR map on the corresponding vector state is simply

$$\mathbf{P}_{\hat{U}}[|\psi\rangle] = \frac{\hat{\mathcal{P}}|\psi\rangle}{\sqrt{||\hat{\mathcal{P}}|\psi\rangle||}}. \quad (7.33)$$

As each $\hat{\rho}$ is mapped onto a specific $\mathbf{P}_{\hat{U}}[\hat{\rho}]$ by (7.31), P-CTCs do not suffer from uniqueness ambiguities, unlike D-CTCs. An important point to note however is that because (7.32) is not in general unitary, the theory must be renormalized, which preserves its probabilistic interpretation at the cost of linearity.

7.2.4 Determination of the P-CTC CV state

A striking observation in a comparison of D-CTCs and P-CTCs is the sheer incompatibility between them. Of their many differences, perhaps the most perplexing of all is that one cannot assign a state to the system on the P-CTC [189], while the determinability of the state on the D-CTC is crucial to the success of the theory. For the former, provided all boundary conditions in the path-integral picture [106] are satisfied by the CV state (residing on the P-CTC), then the ability of the P-CTC prescription to resolve any given time-travel paradox is entirely independent of, and agnostic to, this state. The D-CTC model on the other hand relies fundamentally on the CV state being definable, as it is upon this object which the notion of temporal self-consistency is imposed, with the CR output state being directly dependent on it.

Although the state on the P-CTC is ordinarily unassignable, the P-CTC itself is the conduit by which information is necessarily transported to the past. In principle, this means that some CV state exists physically, and so must admit a definite form. It is therefore natural to ponder exactly what state the quantum system on the P-CTC assumes for any given scenario. A methodology which provides the ability to investigate the CV states in quantum prescriptions of time travel would therefore be both useful and enlightening.

To solve this problem, one might think of employing the well-established technique of *quantum state tomography* [1, 20, 193, 194, 195]. This essentially consists of the determination of an unknown quantum state by measuring it with respect to all elements of an appropriate *tomographically* or *informationally* complete basis (set of observables). Examples of such bases include the Pauli matrices (plus the identity matrix) (3.16) for qubits, and the Gell-Mann matrices (also with the identity) (3.22) for qutrits. Once the unknown state (or rather, an ensemble of identically prepared quantum states) has been measured exhaustively in a suitable basis, the resulting statistics (i.e., expectation values) enable the determination of the Bloch vector via (3.25), which is equivalent to precise identification of the associated (and previously unknown) state.

At first glance, this could conceivably allow for the determination of the state of the time-travelling (CV) system in any given quantum prescription of antichronological time travel. A problem with this methodology however is its incompatibility with the requirement of resolving the state without simultaneously disturbing it (i.e., collapsing the wave function). Such a feature is essential, as an ordinary (“strong”) measurement would necessarily disturb the CV state, thereby disrupting the relevant self-consistency condition(s) in any given quantum formulation of CTCs (like D-CTCs and P-CTCs). Therefore, in order to measure the state on the CTC without (significantly) perturbing the system, one can use so-called *weak measurements* [196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213], that is, quantum measurements which minimize disturbance by leaving the measured state unaffected (to first order). This technique and its findings were first presented in [214].

Using this methodology, it is possible to rigorously assign a state to the system on the P-CTC, with such a state taking the form

$$\mathbf{p}_{\hat{U}}[\hat{\rho}] \equiv \text{tr}_{\text{CR}} [\hat{U}(\hat{\rho} \otimes \frac{1}{2}\hat{I})\hat{U}^\dagger]. \quad (7.34)$$

In contrast to its D-CTC counterpart, this state can be computed, and is in fact also unique, for *every* given combination of interaction \hat{U} and input state $\hat{\rho}$. Another interesting observation is that it is exactly equivalent to the CV output state from the first iteration of a maximally mixed seed state in the D-CTC ECP *Iterative picture: The equivalent circuit* (page 65). In essence, this means that a P-CTC can be thought of as being somewhat like a “partial” D-CTC—one in which the self-consistency condition is not fully realized (at least according to D-CTCs). This is curious result, as it suggests that despite their striking differences, the distinct notions of self-consistency that underpin both P-CTCs and D-CTCs may be more similar than originally thought.

An important point to note is that the result (7.34) is applicable only to 2-dimensional (qubit) systems, and that a similar expression for d -dimensional (qudit) systems has yet to be determined analytically. One such candidate however is the natural generalization

$$\mathbf{p}_{\hat{U}}[\hat{\rho}] \equiv \text{tr}_{\text{CR}} [\hat{U}(\hat{\rho} \otimes \frac{1}{d}\hat{I})\hat{U}^\dagger]. \quad (7.35)$$

It is important to reiterate that while this expression may seem perfectly reasonable, its validity as the P-CTC CV state has not been proven. While likely highly non-trivial, it is expected however that such generalization is an entirely possible task. Therefore, an interesting next step would be to study the application of the weak-measurement tomography scheme to d -level (qudit) systems, as such work should provide more insight into the generality of the methodology and its results.

7.3 Comparison of D-CTCs and P-CTCs

In ordinary quantum mechanics, any mixed state can be purified in an enlarged Hilbert space through the use of an ancillary system. In the Deutsch model however, purification of the CV state is not possible since it has to be in a proper mixture (that is, it can not be a subsystem of a pure entangled state) [215]. The very power of the prescription lies in the fact that it allows for the information trapped within the CTC to exist in a mixed state, and it is this distinguishing feature which enables it to solve classically paradoxical scenarios.

P-CTCs on the other hand provide solutions by ignoring the precise mechanism behind antichronological time travel. Instead, the theory posits that the effect is simply mathematically equivalent to the operational protocol of preselecting and postselecting against the same maximally entangled state.

Evidently, these two prescriptions are distinct, yet they each resolve time-travel paradoxes, albeit in usually very different ways. While P-CTCs provide unique resolutions, D-CTCs alternatively require an additional condition (see [23, 76, 185, 186, 187, 188]) to arrive at a unique solution. However, since these prescriptions are treatments of *interacting* quantum systems near CTCs, we observe non-unitarity in the evolutions of CR systems in both models, which in turn means that the output states are non-linear functions of the input state(s). In the case of P-CTCs, the output states are unnormalized (thereby requiring renormalization), and such an effect jeopardizes the usual probabilistic interpretation of quantum states. The non-linearity and non-unitarity of D-CTCs on the other hand is manifestly visible in the production of entropy through mixing of the input states. This is clear from its equations of motion (7.4) and (7.5), which essentially describe an open interaction between two quantum systems.

Another important difference between the two prescriptions is that for D-CTCs there is no restriction on the initial data, whilst with P-CTCs, the renormalization leads to limits on the initial data as a function of the future. This aspect of the model means that, for certain interactions, there are forbidden input states, which mathematically is due to destructive interference in the P-CTC path-integral picture. This in turns leads to antichronological and superluminal influence [76, 182, 216, 217], which is contrary to many of the expectations we have of any physical theory that is to be considered “good”.

Non-linearity is a highly non-trivial issue, as its presence fundamentally changes the structure of the theory. This is because the standard proofs of many key theorems and concepts in quantum mechanics, including the no-signalling theorem, the no-cloning theorem, and the inability to distinguish between non-orthogonal states, depend on linearity. Therefore, given the non-linearity and non-unitarity of both D-CTCs and P-CTCs, it unsurprising that these prescriptions possess some powerful features with very interesting consequences.

When used as computational resources, time machines provide significant increases in computational speed and efficiency. According to D-CTCs, both classical and quantum computers with access to a CTC can solve PSPACE-complete [218, 219] problems while a quantum computer can solve NP-complete [218, 220] problems efficiently. Alternatively, P-CTCs tell us that CTC-assisted quantum computers, while being able to solve decision problems efficiently in $\text{NP} \cap \text{co-NP}$ and probabilistically in NP [221], can only generally solve problems in PP [222]. This means that P-CTCs are (putatively) computationally inferior to D-CTCs, though both are still vastly superior to the regular BQP power of quantum computers and BPP power of classical computers.

In addition to their use as computational resources, both D-CTCs [223] and P-CTCs [221] permit non-orthogonal quantum states to be distinguished to some degree. D-CTCs [224, 225] can also clone arbitrary states, and P-CTCs can both signal to the past [216] and delete arbitrary states (as a consequence of being able to solve problems in PP).

7.4 Alternative formulations of quantum time travel

It is evident from the discussion in *Comparison of D-CTCs and P-CTCs* (page 71) that both D-CTCs and P-CTCs possess a myriad of undesirable traits. It is also clear that there is more than one way to extend ordinary quantum theory to include self-consistent antichronological time travel. Thus, one can hope that a quantum prescription of time travel which is an improvement over D-CTCs and P-CTCs may be able to be formulated. At the very least, constructing an alternative theory that is void of at least some of the problems discussed in prior sections should not be an impossible task. With this in mind, it is useful to specify a few of the desired characteristics (based on those in [23]) of such a theory:

1. *Compatibility with standard quantum theory*: The prescription should reproduce ordinary quantum mechanics both along the CTC itself and in regions outside any chronology-violating sets. Compatibility with all experimental evidence of standard quantum mechanics is also desired. This means that the theory should follow all established features of quantum mechanics, such as obeyance of the no-cloning theorem.
2. *Existence and uniqueness of solutions*: The prescription should provide exactly one solution (CR state) for each combination of possible interaction and CV state (if applicable). If multiple such solutions exist, then the probabilities for each should be specified. In other words, the theory should not possess any uniqueness ambiguities like that of D-CTCs.
3. *Absence of pathological traits*: The prescription should be consistent with other generally accepted physical theories such as special relativity (at least in a local, approximate sense) and its associated causal structure. In particular, the theory should not permit *antichronological action*. This is, in essence, the concept where the very *existence* of a time machine in the future, no matter how distant, jeopardizes the agency of an observer in the past and present. For P-CTCs, this arises as restrictions on a system’s state in the past as a function of the future, and constitutes a profoundly interesting (and undesirable) issue for the prescription.

There are of course other traits which may be considered, one such being prejudice in requiring either or both of the CR and CV states to be pure. This characteristic however is often left to the opinions of the physicist, given that there is no objective, *a priori* reasoning as to prefer pure states over mixed states as fundamental objects in any given theory of quantum time travel.

The search for alternatives to D-CTCs and P-CTCs that possess fewer pathological problems is a central theme of research into quantum time travel. Here, we briefly discuss several of the more prominent models (excluding some lesser-developed theories [226, 227]), some of which represent significant departures from the standard method of incorporating the principle of self-consistency into ordinary quantum mechanics.

7.4.1 Weighted D-CTCs (WD-CTCs)

Perhaps the simplest “improved” prescription is that of *weighted D-CTCs* [23], which merely represents an extension of the ordinary D-CTCs model. The premise is simple: associate to each valid D-CTC fixed-point density operator $\hat{\tau}_\alpha$ a weight $w_\alpha \geq 0$, using which the unique, “correct” D-CTC CV state is then given by the integral over the space of all D-CTC fixed points, that is,

$$\hat{\tau} = \frac{\int d\alpha w_\alpha \hat{\tau}_\alpha}{\int d\alpha w_\alpha}. \quad (7.36)$$

The generic meaning of integrating over the state space is to sum over all possibilities (see *Integrals over quantum states* (page 39) for a mathematical definition). The weighted mixture (7.36) therefore represents the combination of all elements in the (convex) subset of valid D-CTC CV states. Accordingly, the uniqueness ambiguity in the ordinary D-CTC prescription is resolved for any given set of weights.

The corresponding weighted D-CTC CR output is therefore given by the usual D-CTC CR map, that is,

$$\begin{aligned} \mathbf{W}_{\hat{U}}[\hat{\rho}, \hat{\tau}] &\equiv \text{tr}_{\text{CV}}[\hat{U}(\hat{\rho} \otimes \hat{\tau})\hat{U}^\dagger] \\ &= \frac{\int d\alpha w_\alpha \text{tr}_{\text{CV}}[\hat{U}(\hat{\rho} \otimes \hat{\tau}_\alpha)\hat{U}^\dagger]}{\int d\alpha w_\alpha}. \end{aligned} \quad (7.37)$$

The parametrization of the constituent states by α evidently describes a whole class of models based on both how the corresponding weights w_α are distributed, and the specific form of the integration measure one chooses to employ. For example, perhaps the simplest model is one in which all weights are equal, i.e., $w_\alpha = 1$. This corresponds to a *uniform* weighted D-CTC theory that is manifestly unique, which immediately is an improvement over standard D-CTCs. More sophisticated models may opt to appoint the weights to represent certain transition probabilities of the CV state through the time-machine region—see [23] for more discussion.

7.4.2 Transition probabilities (T-CTCs)

The idea of quantum transition probabilities can be incorporated into P-CTCs, and the resultant *transition probabilities* prescription (T-CTCs) [23] of quantum time travel shares many features of P-CTCs, while also gaining a few improvements in regards to the pathological problems possessed by the latter. Where a P-CTC preselects and postselects against a maximally entangled state, a T-CTC alternatively uses just a pure state $|\phi\rangle \in \mathcal{P}(\mathcal{H}_{\text{CV}})$ in the CV system. This system is prejudicially assumed to be pure, as a postulate of the theory is that the primitive states of quantum mechanics are similarly pure. Thus, given the CR input $|\psi\rangle \in \mathcal{P}(\mathcal{H}_{\text{CR}})$, the evolution of the bipartite input is

$$|\psi\rangle\langle\psi| \otimes |\phi\rangle\langle\phi| \rightarrow \hat{U}(|\psi\rangle\langle\psi| \otimes |\phi\rangle\langle\phi|)\hat{U}^\dagger \quad (7.38)$$

where \hat{U} describes the (unitary) interaction between the CR and CV modes. Self-consistency of the time-travelling CV state is achieved by postselection on the CV subsystem against the same input state. Doing so yields

$$(\hat{I} \otimes \langle\phi|)\hat{U}(|\psi\rangle\langle\psi| \otimes |\phi\rangle\langle\phi|)\hat{U}^\dagger(\hat{I} \otimes |\phi\rangle) = \hat{U}_\phi|\psi\rangle\langle\psi|\hat{U}_\phi^\dagger \quad (7.39)$$

where

$$\hat{U}_\phi \equiv (\hat{I} \otimes \langle\phi|)\hat{U}(\hat{I} \otimes |\phi\rangle) \quad (7.40)$$

is an operator which acts on \mathcal{H}_{CR} and is not unitary in general. The CR output state $\mathbf{T}_{\hat{U}}[\psi]$ is then obtained when one integrates over all such CV states and subsequently renormalizes, i.e.,

$$\begin{aligned} \mathbf{T}_{\hat{U}}[\psi] &= \frac{1}{\mathcal{N}} \int_{\mathcal{P}(\mathcal{H}_{\text{CV}})} d[\phi] \hat{U}_\phi |\psi\rangle\langle\psi|\hat{U}_\phi^\dagger, \\ \mathcal{N} &\equiv \int_{\mathcal{P}(\mathcal{H}_{\text{CV}})} d[\phi] \langle\psi|\hat{U}_\phi \hat{U}_\phi^\dagger |\psi\rangle. \end{aligned} \quad (7.41)$$

Although the expression (7.41) is concise, it is difficult to ascertain the behaviour of the CR output state given the integral form. However, following the original work in [23], the form of (7.41) may be simplified and written in terms of only states and operators. Given an orthonormal basis $\{|\mu\rangle\}_{\mu=0}^{d-1}$ for \mathcal{H}_{CV} , we first expand the unitary interaction \hat{U} in the form

$$\hat{U} = \sum_{\alpha,\beta=0}^{d-1} \hat{V}_{\alpha\beta} \otimes |\alpha\rangle\langle\beta| \quad (7.42)$$

where $\hat{V}_{\alpha\beta}$ are operators on \mathcal{H}_{CR} . Using this and the integrals

$$I_{\alpha\beta,\gamma\delta} \equiv \int_{\mathcal{P}(\mathcal{H}_{\text{CV}})} d[\phi] \langle\phi|\alpha\rangle\langle\beta|\phi\rangle\langle\phi|\delta\rangle\langle\gamma|\phi\rangle \quad (7.43)$$

we may express (7.41) as

$$\mathbf{T}_{\hat{U}}[\psi] = \frac{1}{N} \sum_{\alpha,\beta,\gamma,\delta=0}^{d-1} I_{\alpha\beta,\gamma\delta} \hat{V}_{\alpha\beta} |\psi\rangle\langle\psi| \hat{V}_{\gamma\delta}^\dagger. \quad (7.44)$$

In the integral (7.43), we expand both the state $|\phi\rangle$ and measure $d[\phi]$ using the Hurwitz parametrization [(3.81) and (3.82), respectively]. For each $|\mu\rangle$, the $d\varphi_\mu$ component of the corresponding measure factors out, and so any integrand whose φ_μ -dependence is only an integer power of $e^{i\varphi_\mu}$ necessarily integrates to zero due to the identity

$$\int_{\varphi=0}^{2\pi} d\varphi e^{i(m-n)\varphi} = 2\pi\delta_{mn}, \quad m \neq n, \quad m, n \in \mathbb{Z}. \quad (7.45)$$

This means that any integral (7.43) vanishes unless it is of the form,

$$I_{\alpha\alpha,\beta\beta} = I_{\alpha\beta,\alpha\beta} = \int_{\mathcal{P}(\mathcal{H}_{\text{CV}})} d[\phi] |\langle\alpha|\phi\rangle|^2 |\langle\beta|\phi\rangle|^2, \quad (7.46)$$

and so the only surviving terms in the expression (7.44) are

$$\mathbf{T}_{\hat{U}}[\psi] = \frac{1}{N} \left(\sum_{\substack{\alpha,\beta=0 \\ \alpha \neq \beta}}^{d-1} I_{\alpha\beta,\alpha\beta} \hat{V}_{\alpha\beta} |\psi\rangle\langle\psi| \hat{V}_{\alpha\beta}^\dagger + \sum_{\substack{\alpha,\beta=0 \\ \alpha \neq \beta}}^{d-1} I_{\alpha\alpha,\beta\beta} \hat{V}_{\alpha\alpha} |\psi\rangle\langle\psi| \hat{V}_{\beta\beta}^\dagger + \sum_{\alpha=0}^{d-1} I_{\alpha\alpha,\alpha\alpha} \hat{V}_{\alpha\alpha} |\psi\rangle\langle\psi| \hat{V}_{\alpha\alpha}^\dagger \right). \quad (7.47)$$

The usefulness of having a parametrization for which the integration measure is invariant under unitary transformations now comes to the fore: one may rotate the state $|\phi\rangle$ in the integrals (7.46) so that only the $|d-1\rangle$ and $|d-2\rangle$ components contribute. Therefore, under the Hurwitz parametrization, we find for $\alpha \neq \beta$,

$$\begin{aligned} I_{\alpha\beta,\alpha\beta} &= I_{\alpha\alpha,\beta\beta} = \int_{\mathcal{P}(\mathcal{H}_{\text{CV}})} d[\phi] |\langle d-1|\phi\rangle|^2 |\langle d-2|\phi\rangle|^2 \\ &= (2\pi)^{d-1} \prod_{\gamma=1}^{d-3} \int_{\vartheta_\gamma \in [0, \frac{\pi}{2}]} d\vartheta_\gamma \cos(\vartheta_\gamma) \sin^{2\gamma-1}(\vartheta_\gamma) \\ &\quad \times \int_{\vartheta_{d-2} \in [0, 2\pi]} d\vartheta_{d-2} \cos^3(\vartheta_{d-2}) \sin^{2d-5}(\vartheta_{d-2}) \\ &\quad \times \int_{\vartheta_{d-1} \in [0, 2\pi]} d\vartheta_{d-1} \cos^3(\vartheta_{d-1}) \sin^{2d-1}(\vartheta_{d-1}), \end{aligned} \quad (7.48)$$

while for $\alpha = \beta$,

$$\begin{aligned} I_{\alpha\alpha,\alpha\alpha} &= \int_{\mathcal{P}(\mathcal{H}_{\text{CV}})} d[\phi] |\langle d-1|\phi\rangle|^4 \\ &= (2\pi)^{d-1} \prod_{\gamma=1}^{d-3} \int_{\vartheta_\gamma \in [0, \frac{\pi}{2}]} d\vartheta_\gamma \cos(\vartheta_\gamma) \sin^{2\gamma-1}(\vartheta_\gamma) \\ &\quad \times \int_{\vartheta_{d-2} \in [0, 2\pi]} d\vartheta_{d-2} \cos(\vartheta_{d-2}) \sin^{2d-5}(\vartheta_{d-2}) \\ &\quad \times \int_{\vartheta_{d-1} \in [0, 2\pi]} d\vartheta_{d-1} \cos^5(\vartheta_{d-1}) \sin^{2d-3}(\vartheta_{d-1}). \end{aligned} \quad (7.49)$$

Evaluation (numerical or otherwise) of the integrals in these forms reveals that

$$\frac{I_{\alpha\alpha,\alpha\alpha}}{I_{\alpha\beta,\alpha\beta}} = \frac{I_{\alpha\alpha,\alpha\alpha}}{I_{\alpha\alpha,\beta\beta}} = 2, \quad \alpha \neq \beta, \quad (7.50)$$

which allows us to combine the distinct sums in (7.47) and bring out a common multiplicative pre-factor (which we subsequently neglect due to the foresight of having to renormalize), yielding

$$\mathbf{T}_{\hat{U}}[\psi] \propto \sum_{\alpha,\beta=0}^{d-1} \left(\hat{V}_{\alpha\beta} |\psi\rangle\langle\psi| \hat{V}_{\alpha\beta}^\dagger + \hat{V}_{\alpha\alpha} |\psi\rangle\langle\psi| \hat{V}_{\beta\beta}^\dagger \right). \quad (7.51)$$

Finally, with the identities

$$\begin{aligned} \sum_{\alpha=0}^{d-1} \hat{V}_{\alpha\alpha} &= \text{tr}_{\text{CV}}[\hat{U}], \\ \sum_{\alpha,\beta=0}^{d-1} \hat{V}_{\alpha\beta} |\psi\rangle\langle\psi| \hat{V}_{\alpha\beta}^\dagger &= \text{tr}_{\text{CV}} [\hat{U}(|\psi\rangle\langle\psi| \otimes \hat{I}) \hat{U}^\dagger], \end{aligned} \quad (7.52)$$

we arrive at the result

$$\begin{aligned} \mathbf{T}_{\hat{U}}[\psi] &= \frac{1}{N'} \left(\text{tr}_{\text{CV}}[\hat{U}] |\psi\rangle\langle\psi| \text{tr}_{\text{CV}}[\hat{U}^\dagger] + d \text{tr}_{\text{CV}} [\hat{U}(|\psi\rangle\langle\psi| \otimes \frac{1}{d}\hat{I}) \hat{U}^\dagger] \right), \\ N' &\equiv \text{tr} \left\{ \text{tr}_{\text{CV}}[\hat{U}] |\psi\rangle\langle\psi| \text{tr}_{\text{CV}}[\hat{U}^\dagger] \right\} + d. \end{aligned} \quad (7.53)$$

In general, the methodology of the T-CTC prescription may equally be applied to mixed CR input states, giving

$$\begin{aligned} \mathbf{T}_{\hat{U}}[\hat{\rho}] &= \frac{1}{N'} \left(\text{tr}_{\text{CV}}[\hat{U}] \hat{\rho} \text{tr}_{\text{CV}}[\hat{U}^\dagger] + d \text{tr}_{\text{CV}} [\hat{U}(\hat{\rho} \otimes \frac{1}{d}\hat{I}) \hat{U}^\dagger] \right), \\ N' &\equiv \text{tr} \left\{ \text{tr}_{\text{CV}}[\hat{U}] \hat{\rho} \text{tr}_{\text{CV}}[\hat{U}^\dagger] \right\} + d. \end{aligned} \quad (7.54)$$

In this form, it is immediately clear that the T-CTC equation of motion is a weighted mixture of the P-CTC equation of motion (7.31) with an ordinary quantum channel, giving the impression that a T-CTC can be thought of as a noisy P-CTC. Therefore, T-CTCs, like P-CTCs, does not suffer from uniqueness ambiguities, while having the benefit of being consistent for any given \hat{U} and $\hat{\rho}$. This is because the condition

$$\text{tr} [\text{tr}_{\text{CV}}[\hat{U}] \hat{\rho} \text{tr}_{\text{CV}}[\hat{U}^\dagger]] = 0, \quad (7.55)$$

while rendering P-CTCs inconsistent, poses no problems for T-CTCs. The prescription thus provides unique solutions to all paradoxes with the added advantage of not requiring either an adjunct condition or the introduction of noise in order to do so.

7.4.3 CTC as a ring resonator

In optics, a *ring resonator* involves a scattering process wherein a portion of the output is supplied back into the input. Czachor [228] posits that the same formal structure is encountered in any looped quantum evolution, including time travel to the past (in the neighbourhood of a CTC). Given that the quantum theory of ring resonators is both well-developed and agrees closely with experiment, then it is therefore natural to treat the CTC as a ring resonator, and this provides an alternative prescription with which self-consistent solutions to quantum time travel problems can be formulated.

Czachor's scheme [228] is in essence a topological one, and is thus independent of any spacetime geometry. As a theory of quantum time travel, it provides logically consistent solutions to any given time-travel paradox, with such solutions notably being inequivalent to those given by D-CTCs and P-CTCs. Indeed, while D-CTCs is characterized by an interaction between a time traveller and a copy of themselves on a singly looped CTC (meaning that at any given global time there can physically exist at most two time travellers), the ring resonator paradigm describes the time traveller as a single object that is in a state

of superposition of being inside and outside the loop. Mathematically, the solutions in essence describe the ordinary quantum optical reflection of an input state off of the time machine. This is to say that the interaction with the CTC is in fact merely a self-interference effect, which is an inherently linear phenomenon. The result is a robust, linear, fully self-consistent prescription of quantum time travel that is free from many of the issues which plague D-CTCs and P-CTCs, including violation of the no-cloning theorem. It remains to be studied however whether this model possesses any advantages in the context of quantum information processing.

7.4.4 Indefinite causal structure and the process matrix formalism

Under a complete theory of quantum gravity, it is expected that spacetime loses its classical properties [229, 230]. One of the possible consequences of this is the emergence of indefinite causal structures [146, 148, 231], where the causal order of events is no longer definite and instead is in a quantum (probabilistic) superposition of all possible orderings. However, perhaps the foremost problem that arises when one abandons classical causality is the appearance of logical inconsistencies, themselves stemming from seemingly paradoxical physical scenarios. The usual procedure to eliminate these and restore self-consistency is to extend ordinary quantum mechanics (like in P-CTCs or D-CTCs), as discussed in the preceding sections.

A radically different approach involves the *process matrix formalism*. This theory provides an abstract framework for quantum mechanics on indefinite causal structures, and includes the study of which types of global processes are compatible with the assumption that the standard laws of physics are valid in a local sense [152, 155, 156, 161, 162, 163, 164, 165, 166, 167, 170, 173, 175, 232, 233, 234]. In other words, this formalism posits that all operations which are possible in ordinary spacetime are still allowed in local regions of any spacetime, regardless of the nature of the underlying causality. It is useful as a tool to study correlations between parties where the causal order among them is not specified *a priori*. Instead, the assumption that the probabilities of measurement outcomes are well-defined replaces the (often implicit) assumption of a definite causal order. This provides a powerful methodology—one that is in fact more general than (ordinary) quantum theory—as certain correlations that cannot be explained with a predefined causal order of the parties may arise. Interestingly, when applied to classical physics, the process matrix formalism has shown that there exist classical processes which are incompatible with any notion of causal order between events [154, 158, 160].

In [235], Baumeler et al. propose a classical, deterministic version of the formalism as a possible model for CTCs. They consider the most general deterministic dynamics connecting classical degrees of freedom defined on a set of bounded spacetime regions, under the requirement that such dynamics are compatible with *arbitrary* operations performed in the local regions. Baumeler et al. show that this can be realized entirely through reversible interactions, and furthermore that consistency with local operations is compatible with non-trivial time travel. In particular, they found a simple characterization for all processes involving up to three regions. In other words, it is guaranteed that at least three parties are free to perform arbitrary local operations while interacting in such a way that all are both in the future and in the past of each other.

Tobar and Costa [236] extend the characterization of deterministic processes to an arbitrary number of regions. Central to their results is the conclusion that CTCs are compatible not only with both determinism and a notion of freedom of choice regarding local operations, but also with a wide variety of dynamical processes and scenarios. This means that the way in which CTCs allow multiple observers in distinct regions to communicate with each other is not overly restricted by conflicts between locality, logical consistency, and freedom of choice. Given that this conclusion was reached using an abstract framework that is independent of both particular dynamics and geometry, then this hints at a deep, robust compatibility between time travel and the laws of physics, which is contrary to the previously discussed information-theoretic models of quantum time travel.

Part II

Documentation

In this part, detailed documentation of the Qhronology package is presented. This includes in-depth descriptions of the package's various submodules and their respective functions and classes (including all methods and properties). Numerous usage examples of many of the most important objects are also provided. Note however that only the components of the package which are intended to be used by the end user are included here. For the other, undocumented modules, please see the relevant parts of the package's source code.

Quantum

The `quantum` subpackage contains most of Qhrontology's underlying mathematical framework.

Table 7.1: Overview of Qhrontology's `quantum` subpackage

Module	Contents	Objects
<code>states.py</code>	Classes for the creation of quantum states.	Main class: <code>QuantumState</code> (page 83) Subclasses: <code>VectorState</code> (page 103) <code>MatrixState</code> (page 103) <code>PureState</code> (page 104) <code>MixedState</code> (page 104)
<code>gates.py</code>	Classes for the creation of quantum gates.	Main class: <code>QuantumGate</code> (page 105) Subclasses: <code>Pauli</code> (page 112) <code>GellMann</code> (page 115) <code>Rotation</code> (page 116) <code>Phase</code> (page 117) <code>Diagonal</code> (page 119) <code>Swap</code> (page 121) <code>Summation</code> (page 124) <code>Not</code> (page 125) <code>Hadamard</code> (page 127) <code>Fourier</code> (page 129) <code>Measurement</code> (page 131) Combining gates: <code>GateInterleave</code> (page 133) <code>GateStack</code> (page 135)
<code>circuits.py</code>	A class for the creation of quantum circuits.	Class: <code>QuantumCircuit</code> (page 139)
<code>prescriptions.py</code>	A class for the creation of quantum circuits containing closed timelike curves. Classes and functions implementing quantum prescriptions of time travel.	Main class: <code>QuantumCTC</code> (page 157) Subclasses: <code>DCTC</code> (page 164) <code>PCTC</code> (page 169) Functions: <code>dctc_violating()</code> (page 163) <code>dctc_respecting()</code> (page 163) <code>pctc_violating()</code> (page 168) <code>pctc_respecting()</code> (page 169)

Mechanics

The `mechanics` subpackage contains Qhronology's core logic for creating quantum vectors and matrices, performing operations on such constructs, and computing various significant scalar quantities.

Table 7.2: Overview of Qhronology's `mechanics` subpackage

Module	Contents	Objects
<code>matrices.py</code>	Core functions for creating quantum vectors and matrices.	Functions: <code>vector_basis()</code> (page 175) <code>ket()</code> (page 175) <code>bra()</code> (page 176) <code>quantum_state()</code> (page 177) <code>encode()</code> (page 178) <code>decode_slow()</code> (page 179) <code>decode()</code> (page 180) <code>decode_fast()</code> (page 180) <code>decode_multiple()</code> (page 181)
<code>quantities.py</code>	Functions for computing quantum quantities from matrices. A mixin for endowing compatible classes with the ability to calculate these quantities.	Functions: <code>trace()</code> (page 183) <code>purity()</code> (page 183) <code>distance()</code> (page 184) <code>fidelity()</code> (page 184) <code>entropy()</code> (page 185) <code>mutual()</code> (page 186) Mixin: <code>QuantitiesMixin</code> (page 187)
<code>operations.py</code>	Functions for performing quantum operations on matrices. A mixin for endowing compatible classes with the ability to perform these operations.	Functions: <code>densify()</code> (page 189) <code>columnify()</code> (page 189) <code>dagger()</code> (page 189) <code>simplify()</code> (page 190) <code>apply()</code> (page 190) <code>rewrite()</code> (page 191) <code>normalize()</code> (page 192) <code>coefficient()</code> (page 192) <code>partial_trace()</code> (page 193) <code>measure()</code> (page 194) <code>postselect()</code> (page 196) Mixin: <code>OperationsMixin</code> (page 197)

Structure

For reference, a structure diagram detailing the relationships between Qhrontology's classes in the standard UML (Unified Modelling Language) framework is depicted in [Figure 7.4](#).

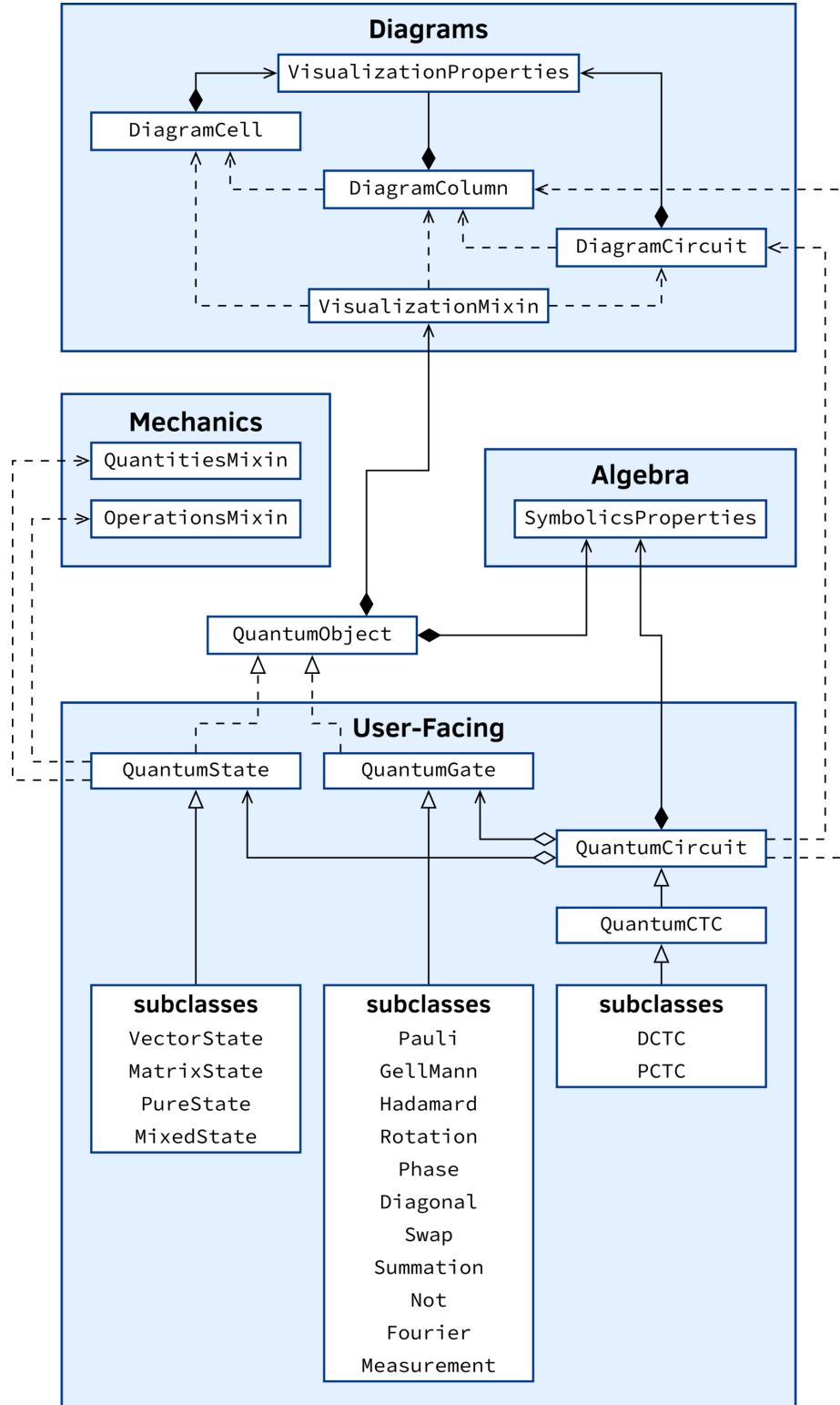


Figure 7.4: The (inheritance) relationships between the package's classes.

Types

Qhronology's underlying functionality takes advantage of a few bespoke type aliases, which are detailed in the table below.

Table 7.3: Internal type aliases

Type Alias	Description	Definition
<code>num</code>	Scalar numerical numbers	<code>numbers.Number numpy.generic sympy.Basic</code>
<code>sym</code>	SymPy symbolic scalar expressions and symbols	<code>sympy.matrices.expressions.matexpr.MatrixSymbol sympy.matrices.expressions.matexpr.MatrixElement sympy.core.symbol.Symbol</code>
<code>mat</code>	SymPy (mutable) dense matrices	<code>sympy.matrices.dense.MutableDenseMatrix</code>
<code>arr</code>	NumPy arrays	<code>numpy.ndarray</code>

In Qhronology, quantum states are both represented and constructed natively in the *computational* basis (also known as the *standard* or *z*-basis). This is achieved primarily through the use of the [QuantumState](#) (page 83) class,

```
from qhronology.quantum.states import QuantumState
```

which itself relies chiefly on functionality provided by the matrix-generating [quantum_state\(\)](#) (page 177) function:

```
from qhronology.mechanics.matrices import quantum_state
```

Characterization of quantum states is facilitated by two properties: `form` distinguishes between states which are "vector" or "matrix", while `kind` distinguishes those which are "pure" or "mixed". Of the four combinations (pairs) of these properties, all are valid except for the pairing of "vector" and "mixed". Therefore, to expedite and simplify state instantiation, the following subclasses of the base class [QuantumState](#) (page 83) are provided:

```
from qhronology.quantum.states import VectorState, MatrixState, PureState, MixedState
```

These classes are *specialized* (or *restrictive*) subclasses, meaning that they do not extend the base class in any way, and instead merely constrain its functionality in order to enforce the desired behaviour. They therefore allow for quantum state objects to be initialized in ways that are more concise than the general [QuantumState](#) (page 83) class.

8.1 Main class

```
class QuantumState(
    spec: MutableDenseMatrix | ndarray | list[list[Number | generic | Basic |
        MatrixSymbol | MatrixElement | Symbol | str]] | list[tuple[Number | generic |
        Basic | MatrixSymbol | MatrixElement | Symbol | str, int | list[int]]],
    form: str | None = None,
    kind: str | None = None,
    dim: int | None = None,
    symbols: dict[MatrixSymbol | MatrixElement | Symbol | str, dict[str, Any]] |
        None = None,
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement |
        Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement |
        Symbol | str]] | None = None,
    conjugate: bool | None = None,
    norm: bool | Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol |
        str | None = None,
    label: str | None = None,
    notation: str | None = None,
    family: str | None = None,
    debug: bool | None = None,
)
```

Bases: [QuantitiesMixin](#) (page 187), [OperationsMixin](#) (page 197), [QuantumObject](#)

A class for creating quantum states and storing their metadata.

Instances provide complete descriptions of both vector and matrix quantum states, along with various associated attributes (such as mathematical conditions, including normalization). The

internal state of the class is expressly mutable, and a selection of useful methods are provided with which the state can be manipulated and otherwise transformed in various quantum-mechanically significant ways. This includes:

- ▶ normalization
- ▶ (partial) trace
- ▶ measurement
- ▶ postselection

Parameters

- ▶ **spec** – The specification of the quantum state. Provides a complete description of the state's values in a standard `dim`-dimensional basis. Can be one of:
 - a SymPy matrix (`mat`)
 - a NumPy array (`arr`)
 - a list of lists of numerical, symbolic, or string expressions (that collectively specify a matrix) (`list[list[num | sym | str]]`)
 - a list of 2-tuples of numerical, symbolic, or string coefficients and their respective number-basis specifications (`list[tuple[num | sym | str, int | list[int]]]`)
- ▶ **form (str)** – A string specifying the *form* for the quantum state to take. Can be either of "`vector`" or "`matrix`". Defaults to "`matrix`".
- ▶ **kind (str)** – A string specifying the *kind* for the quantum state to take. Can be either of "`mixed`" or "`pure`". Defaults to "`mixed`".
- ▶ **dim (int)** – The dimensionality of the quantum state's Hilbert space. Must be a non-negative integer. Defaults to `2`.
- ▶ **symbols (dict[sym | str, dict[str, Any]])** – A dictionary in which the keys are individual symbols (usually found within the state specification `spec`) and the values are dictionaries of their respective SymPy keyword-argument `assumptions`. Defaults to `{}`.
- ▶ **conditions (list[tuple[num | sym | str, num | sym | str]])** – A list of 2-tuples of conditions to be applied to the state. All instances of the expression in each tuple's first element are replaced by the expression in the respective second element. This uses the same format as the SymPy `subs()` method. The order in which they are applied is simply their order in the list. Defaults to `[]`.
- ▶ **conjugate (bool)** – Whether to perform Hermitian conjugation on the state when it is called. Defaults to `False`.
- ▶ **norm (bool | num | sym | str)** – The value to which the state is normalized. If `True`, normalizes to a value of 1. If `False`, does not normalize. Defaults to `False`.
- ▶ **label (str)** – The unformatted string used to represent the state in mathematical expressions. Must have a non-zero length. Defaults to " `ρ` " (if `form == "matrix"`) or " `ψ` " (if `form == "vector"`).
- ▶ **notation (str)** – The formatted string used to represent the state in mathematical expressions. When not `None`, overrides the value passed to `label`. Must have a non-zero length. Not intended to be set by the user in most cases. Defaults to `None`.
- ▶ **family (str)** – A string expressing the kind of block element for which the object is to be visualized. Not intended to be set by the user. Defaults to "`LSTICK`".
- ▶ **debug (bool)** – Whether to print the internal state (held in `matrix`) on change. Defaults to `False`.

Examples

```
>>> qubit_vector = QuantumState(
...     spec=[("a", [0]), ("b", [1])],
...     form="vector",
...     symbols={"a": {"complex": True}, "b": {"complex": True}},
...     conditions=[("a*conjugate(a) + b*conjugate(b)", "1")],
...     norm=1,
...     label="\psi",
... )
>>> qubit_vector.output()
Matrix([
[a],
[b])
>>> qubit_vector.print()
|\psi\rangle = a|0\rangle + b|1\rangle
>>> qubit_vector.diagram()
```

$|\psi\rangle =$

```
>>> qutrit_vector = QuantumState(
...     spec=[("a", [0]), ("b", [1]), ("c", [2])],
...     form="vector",
...     dim=3,
...     symbols={
...         "a": {"complex": True},
...         "b": {"complex": True},
...         "c": {"complex": True},
...     },
...     conditions=[("a*conjugate(a) + b*conjugate(b) + c*conjugate(c)", "1")],
...     norm=1,
...     label="\phi",
... )
>>> qutrit_vector.output()
Matrix([
[a],
[b],
[c])
>>> qutrit_vector.print()
|\phi\rangle = a|0\rangle + b|1\rangle + c|2\rangle
>>> qutrit_vector.diagram()
```

$|\phi\rangle =$

```
>>> qubit_pure = QuantumState(
...     spec=[("alpha", [0]), ("beta", [1])],
...     form="matrix",
...     kind="pure",
...     symbols={"a": {"complex": True}, "b": {"complex": True}},
...     conditions=[("alpha*conjugate(alpha) + beta*conjugate(beta)", 1)],
...     norm=1,
```

(continues on next page)

(continued from previous page)

```

...      label="ξ",
...
>>> qubit_pure.output()
Matrix([
[α*conjugate(α), α*conjugate(β)],
[β*conjugate(α), β*conjugate(β)]])
>>> qubit_pure.print()
|ξ⟩⟨ξ| = α*conjugate(α)|0⟩⟨0| + α*conjugate(β)|0⟩⟨1| + β*conjugate(α)|1⟩⟨0| +
→β*conjugate(β)|1⟩⟨1|
>>> qubit_pure.diagram()

```

 $|\xi\rangle\langle\xi|$ —

```

>>> qubit_mixed = QuantumState(
...     spec=[("p", [0]), ("1 - p", [1])],
...     form="matrix",
...     kind="mixed",
...     symbols={"p": {"real": True}},
...     norm=1,
...     label="τ",
... )
>>> qubit_mixed.output()
Matrix([
[p, 0],
[0, 1 - p]])
>>> qubit_mixed.print()
τ = p|0⟩⟨0| + (1 - p)|1⟩⟨1|
>>> qubit_mixed.diagram()

```

 τ —

```

>>> custom_vector = QuantumState(spec=[[μ], [ν]], kind="mixed", label="η")
>>> custom_vector.output()
Matrix([
[μ*conjugate(μ), μ*conjugate(ν)],
[ν*conjugate(μ), ν*conjugate(ν)]])
>>> custom_vector.print()
η = μ*conjugate(μ)|0⟩⟨0| + μ*conjugate(ν)|0⟩⟨1| + ν*conjugate(μ)|1⟩⟨0| +
→ν*conjugate(ν)|1⟩⟨1|
>>> custom_vector.diagram()

```

 η —

```

>>> custom_matrix = QuantumState(spec=[[w, x], [y, z]], kind="mixed",
...                                label="ω")
>>> custom_matrix.output()

```

(continues on next page)

(continued from previous page)

```
Matrix([
[w, x],
[y, z])
>>> custom_matrix.print()
w = w|0⟩⟨0| + x|0⟩⟨1| + y|1⟩⟨0| + z|1⟩⟨1|
>>> custom_matrix.diagram()
```

 ω —

```
>>> bell_state = QuantumState(spec=[(1, [0, 0]), (1, [1, 1])], form="vector",
... norm=1, label="Φ")
>>> bell_state.output()
Matrix([
[sqrt(2)/2],
[      0],
[      0],
[sqrt(2)/2]])
>>> bell_state.print()
|Φ⟩ = sqrt(2)/2|0,0⟩ + sqrt(2)/2|1,1⟩
>>> bell_state.diagram()
```

 $|Φ\rangle$ —

```
>>> tripartite_zero = QuantumState(spec=[(1, [0, 0, 0])], form="vector", label="0,
... 0,0")
>>> tripartite_zero.output()
Matrix([
[1],
[0],
[0],
[0],
[0],
[0],
[0],
[0]])
>>> tripartite_zero.print()
|0,0,0⟩ = |0,0,0⟩
>>> tripartite_zero.diagram()
```

 $|0,0,0\rangle$ —

8.1.1 Constructor argument properties

```
property QuantumState.spec: MutableDenseMatrix | ndarray | list[list[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, int | list[int]]]
```

The matrix representation of the quantum state. Provides a complete description of the state in a standard `dim`-dimensional basis.

```
property QuantumState.form: str
```

The *form* of the object. Can be either of "vector" or "matrix". Only `QuantumState` (page 83) objects can be "vector".

```
property QuantumState.kind: str
```

The *kind* of quantum state. Can be either of "mixed" or "pure".

```
property QuantumState.dim: int
```

The dimensionality of the quantum object. Must be a non-negative integer.

```
property QuantumState.symbols: dict[MatrixSymbol | MatrixElement | Symbol | str, dict[str, Any]]
```

A dictionary in which the keys are individual symbols (contained within the object's matrix representation) and the values are dictionaries of their respective SymPy keyword-argument `assumptions` ("predicates"). A full list of currently supported predicates, and their defaults, is as follows:

- ▶ "algebraic": True
- ▶ "commutative": True
- ▶ "complex": True
- ▶ "extended_negative": False
- ▶ "extended_nonnegative": True
- ▶ "extended_nonpositive": False
- ▶ "extended_nonzero": True
- ▶ "extended_positive": True
- ▶ "extended_real": True
- ▶ "finite": True
- ▶ "hermitian": True
- ▶ "imaginary": False
- ▶ "infinite": False
- ▶ "integer": True
- ▶ "irrational": False
- ▶ "negative": False
- ▶ "noninteger": False
- ▶ "nonnegative": True
- ▶ "nonpositive": False

- ▶ "nonzero": True
 - ▶ "positive": True
 - ▶ "rational": True
 - ▶ "real": True
 - ▶ "transcendental": False
 - ▶ "zero": False
-

property QuantumState.conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]]

A list of 2-tuples of conditions to be applied to the object's matrix representation.

property QuantumState.norm: bool | Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str

The value to which the state is normalized. If `True`, normalizes to a value of 1. If `False`, does not normalize.

Examples of valid values include:

- ▶ `1/2`
 - ▶ `"1/d"`
 - ▶ `"a*conjugate(a) + b*conjugate(b)"`
-

property QuantumState.conjugate: bool

Whether to perform Hermitian conjugation on the object when it is called.

property QuantumState.label: str

The unformatted string used to represent the object in mathematical expressions. Must have a non-zero length.

property QuantumState.notation: str

The formatted string used to represent the object in mathematical expressions. When set, overrides the value of the `label` property. Must have a non-zero length. Not intended to be set by the user in most cases.

property QuantumState.family: str

The code of the block element that the object is to be visualized as. Not intended to be set by the user.

property QuantumState.debug: bool

Whether to print the object's matrix representation (stored in the `matrix` property) on mutation.

8.1.2 Read-only properties

`property QuantumState.systems: list[int]`

Read-only property containing an ordered list of the numerical indices of the object's systems.

`property QuantumState.num_systems: int`

Read-only property containing the number of systems which the state spans. The current value is calculated from the state's matrix representation and its dimensionality `dim`.

`property QuantumState.is_vector: bool`

Test for whether the object is a vector. Returns `True` if so, otherwise `False`.

`property QuantumState.matrix: MutableDenseMatrix`

The matrix representation of the object.

Considered read-only (this is strictly enforced by `QuantumGate` (page 105) class and its derivatives), though can be (indirectly) mutated by some derived classes (such as `QuantumState` (page 83)). Not intended to be set directly by the user.

8.1.3 Methods

`QuantumState.output(`

```
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
    simplify: bool | None = None,
    conjugate: bool | None = None,
    norm: bool | Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str | None = None,
```

`) → MutableDenseMatrix`

Construct the state's matrix representation, perform any necessary transformations on it, and return it.

Parameters

- ▶ `conditions (list[tuple[num | sym | str, num | sym | str]])` – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ `simplify (bool)` – Whether to perform algebraic simplification on the state. Defaults to `False`.
- ▶ `conjugate (bool)` – Whether to perform Hermitian conjugation on the state. If `False`, does not conjugate. Defaults to the value of `self.conjugate`.
- ▶ `norm (bool | num | sym | str)` – The value to which the state is normalized. If `False`, does not normalize. Defaults to the value of `self.norm`.

Returns

`mat` – The matrix or vector representation of the quantum state.

`QuantumState.diagram(`

```
    pad: tuple[int, int] | None = None,
    sep: tuple[int, int] | None = None,
    style: str | None = None,
    return_string: bool | None = None,
```

`) → None | str`

Print or return a string diagram of the quantum object as a multiline string.

Parameters

- ▶ **pad** (`tuple[int, int]`) – A two-tuple describing the horizontal and vertical interior paddings between the content at the centre of the object (e.g., its label) and its outer edge (e.g., block border). Both integers must be non-negative. Defaults to `(0, 0)`.
- ▶ **sep** (`tuple[int, int]`) – A two-tuple describing the horizontal and vertical exterior separation distances at the object's edges. Both integers must be non-negative. Defaults to `(1, 1)`.
- ▶ **style** (`str`) – A string specifying the style for the circuit visualization to take. Can be any of `"ascii"`, `"unicode"`, or `"unicode_alt"`. Defaults to `"unicode"`.
- ▶ **return_string** (`bool`) – Whether to return the assembled diagram as a multiline string. Defaults to `False`.

Returns

- ▶ `None` – Returned only if `return_string` is `False`.
- ▶ `str` – The rendered circuit diagram. Returned only if `return_string` is `True`.

Note:

The quality of the visualization depends greatly on the output's configuration. For best results, the terminal should have a monospace font with good Unicode coverage.

Examples

For usage examples, please see those of the `QuantumState` (page 83) class itself.

8.1.4 Operations

All of these methods (except for `reset()` (page 91)) are inherited from `OperationsMixin` (page 197).

`QuantumState.reset()`

Reset the quantum state's internal matrix state (specifically its `matrix` property) to its original value at instantiation.

Note:

This reset only the `matrix` property of the instance. All other attributes and properties are unchanged.

`QuantumState.densify()`

Convert the state to its equivalent (density) matrix representation.

States that are already in density matrix form are unmodified.

Examples

```
>>> psi = QuantumState(spec=[("a", [0]), ("b", [1])], form="vector", label="ψ")
>>> psi.print()
|ψ⟩ = a|0⟩ + b|1⟩
>>> psi.densify()
```

(continues on next page)

(continued from previous page)

```
>>> psi.print()
|ψ⟩⟨ψ| = a*conjugate(a)|0⟩⟨0| + a*conjugate(b)|0⟩⟨1| + b*conjugate(a)|1⟩⟨0| +_
- b*conjugate(b)|1⟩⟨1|
```

QuantumState.dagger()

Perform conjugate transposition on the state.

Examples

```
>>> psi = QuantumState(spec=[("a", [0]), ("b", [1])], form="vector", label="ψ")
>>> psi.print()
|ψ⟩ = a|0⟩ + b|1⟩
>>> psi.dagger()
>>> psi.print()
⟨ψ| = conjugate(a)⟨0| + conjugate(b)⟨1|
```

QuantumState.simplify()

Apply a forced simplification to the state using the values of its `symbols` and `conditions` properties.

Useful if intermediate simplification is required during a sequence of mutating operations in order to process the state into a more desirable form.

```
QuantumState.apply(
    function: Callable,
    arguments: dict[str, Any] | None = None,
)
```

Apply a Python function (`function`) to the state.

Useful when used with SymPy's symbolic-manipulation functions, such as:

- ▶ `simplify()`
- ▶ `expand()`
- ▶ `factor()`
- ▶ `collect()`
- ▶ `cancel()`
- ▶ `apart()`

More can be found at:

- ▶ [SymPy documentation: Simplification²¹](#)
- ▶ [SymPy documentation: Simplify²²](#)

Parameters

- ▶ **`function` (`Callable`)** – A Python function. Its first non-keyword argument must be able to take a mathematical expression or a matrix/array of such types.
- ▶ **`arguments` (`dict[str, str]`)** – A dictionary of keyword arguments (both keys and values as strings) to pass to the `function` call. Defaults to `{}`.

Examples

```
>>> psi = QuantumState(
...     spec=[("a*b + b*c + c*a", [0]), ("x*y + y*z + z*x", [1])], form="vector",
...     label="ψ"
... )
>>> psi.print()
|ψ⟩ = (a*b + a*c + b*c)|0⟩ + (x*y + x*z + y*z)|1⟩
>>> psi.apply(sp.collect, {"syms": ["a", "x"]})
>>> psi.print()
|ψ⟩ = (a*(b + c) + b*c)|0⟩ + (x*(y + z) + y*z)|1⟩
>>> psi.apply(sp.expand)
>>> psi.print()
|ψ⟩ = (a*b + a*c + b*c)|0⟩ + (x*y + x*z + y*z)|1⟩
```

`QuantumState.rewrite(
 function: Callable,
)`

Rewrite the elements of the state using the given mathematical function (`function`).

Useful when used with SymPy's mathematical functions, such as:

- ▶ `exp()`
- ▶ `log()`
- ▶ `sin()`
- ▶ `cos()`

Parameters

`function (Callable)` – A SymPy mathematical function.

Examples

```
>>> psi = QuantumState(spec=[("cos(θ)", [0]), ("sin(θ)", [1])], form="vector",
...     label="ψ")
>>> psi.print()
|ψ⟩ = cos(θ)|0⟩ + sin(θ)|1⟩
>>> psi.rewrite(sp.exp)
>>> psi.print()
|ψ⟩ = (exp(I*θ)/2 + exp(-I*θ)/2)|0⟩ + -I*(exp(I*θ) - exp(-I*θ))/2|1⟩
```

`QuantumState.normalize(
 norm: Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol |
 str | None = None,
)`

Perform a forced (re)normalization on the state to the value specified (`norm`).

Useful when applied to the current quantum state both before and after mutating operations, prior to any simplification (such as renormalization) performed on the processed output (obtained via the `state()` method).

Parameters

`norm (num | sym | str)` – The value to which the state is normalized. Defaults to `1`.

²¹ <https://docs.sympy.org/latest/tutorials/intro-tutorial/simplification.html>

²² <https://docs.sympy.org/latest/modules/simplify/simplify.html>

Examples

```
>>> identity = QuantumState(spec=[("1", [0]), ("1", [1])], label="I")
>>> identity.print()
I = |0⟩⟨0| + |1⟩⟨1|
>>> identity.normalize()
>>> identity.print()
I = 1/2|0⟩⟨0| + 1/2|1⟩⟨1|
```

```
>>> psi = QuantumState(spec=[("a", [0]), ("b", [1])], form="vector", label="ψ")
>>> psi.print()
|ψ⟩ = a|0⟩ + b|1⟩
>>> psi.normalize()
>>> psi.print()
|ψ⟩ = a/sqrt(a*conjugate(a) + b*conjugate(b))|0⟩ + b/sqrt(a*conjugate(a) + b*conjugate(b))|1⟩
```

QuantumState.coefficient

```
scalar: Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol |
str | None = None,
```

```
)
```

Multiply the state by a scalar value (`scalar`).

Can be useful to manually (re)normalize states, or introduce a phase factor.

Parameters

`scalar` (`num` | `sym` | `str`) – The value by which the state is multiplied. Defaults to `1`.

Examples

```
>>> psi = QuantumState(spec=[("1", [0]), ("1", [1])], form="vector", label="ψ")
>>> psi.print()
|ψ⟩ = |0⟩ + |1⟩
>>> psi.coefficient(1 / sp.sqrt(2))
>>> psi.print()
|ψ⟩ = sqrt(2)/2|0⟩ + sqrt(2)/2|1⟩
```

QuantumState.partial_trace

```
targets: int | list[int] | None = None,
discard: bool | None = None,
optimize: bool | None = None,
```

```
)
```

Perform a partial trace operation on the state.

Performs a total trace if `targets` is unspecified.

Parameters

- ▶ `targets` (`int` | `list[int]`) – The numerical index/indices of the subsystem(s) to be partially traced over. Indexing begins at `0`. Defaults to `[]`.
- ▶ `discard` (`bool`) – Whether the systems corresponding to the indices given in `targets` are to be discarded (`True`) or kept (`False`). Defaults to `True`.
- ▶ `optimize` (`bool`) – Whether to optimize the partial trace implementation's algorithm. Can greatly increase the computational efficiency at the cost of a larger memory footprint during computation. Defaults to `True`.

Examples

```
>>> psi = QuantumState(
...     spec=[("a*u", [0, 0]), ("b*u", [1, 0]), ("a*v", [0, 1]), ("b*v", [1, 1])],
...     form="vector",
...     conditions=[
...         ("a*conjugate(a) + b*conjugate(b)", "1"),
...         ("u*conjugate(u) + v*conjugate(v)", "1"),
...     ],
...     label="Ψ",
... )
>>> psi.print()
|Ψ⟩ = a*u|0,0⟩ + a*v|0,1⟩ + b*u|1,0⟩ + b*v|1,1⟩
>>> psi.partial_trace([1])
>>> psi.simplify()
>>> psi.notation = "ρ"
>>> psi.print()
ρ = a*conjugate(a)|0⟩⟨0| + a*conjugate(b)|0⟩⟨1| + b*conjugate(a)|1⟩⟨0| +_
 -b*conjugate(b)|1⟩⟨1|
```

```
>>> bell = QuantumState(
...     spec=[("1", [0, 0]), ("1", [1, 1])], form="vector", norm=1, label="Φ"
... )
>>> bell.print()
|Φ⟩ = sqrt(2)/2|0,0⟩ + sqrt(2)/2|1,1⟩
>>> bell.partial_trace([0])
>>> bell.notation = "ρ"
>>> bell.print()
ρ = 1/2|0⟩⟨0| + 1/2|1⟩⟨1|
```

`QuantumState.measure()`

```
operators: list[mat | arr | QuantumObject],
targets: int | list[int] | None = None,
observable: bool | None = None,
statistics: bool | None = None,
) → None | list[num | sym]
```

Perform a quantum measurement on one or more systems (indicated in `targets`) of the state.

This method has two main modes of operation:

- When `statistics` is `True`, the (reduced) state ($\hat{\rho}$) (residing on the systems indicated in `targets`) is measured and the set of resulting statistics is returned. This takes the form of an ordered list of values $\{p_i\}_i$ associated with each given operator, where:

- $p_i = \text{tr}[\hat{K}_i^\dagger \hat{K}_i \hat{\rho}]$ (measurement probabilities) when `observable` is `False` (`operators` is a list of Kraus operators or projectors \hat{K}_i)
- $p_i = \text{tr}[\hat{O}_i \hat{\rho}]$ (expectation values) when `observable` is `True` (`operators` is a list of observables \hat{O}_i)

- When `statistics` is `False`, the (reduced) state ($\hat{\rho}'$) (residing on the systems indicated in `targets`) is measured and mutated it according to its predicted post-measurement form (i.e., the sum of all possible measurement outcomes). This yields the transformed states:

- When `observable` is `False`:

$$\hat{\rho}' = \sum_i \hat{K}_i \hat{\rho} \hat{K}_i^\dagger. \quad (8.1)$$

- When `observable` is `True`:

$$\hat{\rho}' = \sum_i \text{tr}[\hat{O}_i \hat{\rho}] \hat{O}_i. \quad (8.2)$$

In the case where `operators` contains only a single item (\hat{K}) and the current state ($|\psi\rangle$) is a vector form, the transformation of the state is in accordance with the rule

$$|\psi'\rangle = \frac{\hat{K}|\psi\rangle}{\sqrt{\langle\psi|\hat{K}^\dagger\hat{K}|\psi\rangle}} \quad (8.3)$$

when `observable` is `False`. In all other mutation cases, the post-measurement state is a matrix, even if the pre-measurement state was a vector.

The items in the list `operators` can also be vectors (e.g., $|\xi_i\rangle$), in which case each is converted into its corresponding operator matrix representation (e.g., $|\xi_i\rangle\langle\xi_i|$) prior to any measurements.

Parameters

- ▶ `operators` (`list[mat | arr | QuantumObject]`) – The operator(s) with which to perform the measurement. These would typically be a (complete) set of Kraus operators forming a POVM, a (complete) set of (orthogonal) projectors forming a PVM, or a set of observables constituting a complete basis for the relevant state space.
- ▶ `targets` (`int | list[int]`) – The numerical indices of the subsystem(s) to be measured. They must be consecutive, and their number must match the number of systems spanned by all given operators. Indexing begins at `0`. All other systems are discarded (traced over) in the course of performing the measurement. Defaults to the value of `self.systems`.
- ▶ `observable` (`bool`) – Whether to treat the items in `operators` as observables instead of Kraus operators or projectors. Defaults to `False`.
- ▶ `statistics` (`bool`) – Whether to return a list of probabilities (`True`) or mutate the state into a post-measurement probabilistic sum of all outcomes (`False`). Defaults to `False`.

Returns

- ▶ `None` – Returned only if `statistics` is `False`.
- ▶ `num / sym / list[num / sym]` – A list of probabilities corresponding to each operator given in `operators`. Returned only if `statistics` is `True`.

Note:

This method does not check for validity of supplied POVMs or the completeness of sets of observables, nor does it renormalize the post-measurement state.

Examples

```
>>> psi = QuantumState(spec=[("a", [0]), ("b", [1])], form="vector", label="ψ")
>>> psi.print()
|ψ⟩ = a|0⟩ + b|1⟩
>>> I = Pauli(index=0)
>>> X = Pauli(index=1)
>>> Y = Pauli(index=2)
>>> Z = Pauli(index=3)
>>> psi.measure(operators=[I, X, Y, Z], observable=True, statistics=True)
[a*conjugate(a) + b*conjugate(b),
 a*conjugate(b) + b*conjugate(a),
 I*(a*conjugate(b) - b*conjugate(a)),
 a*conjugate(a) - b*conjugate(b)]
>>> psi.measure(operators=[I, X, Y, Z], observable=True, statistics=False)
>>> psi.simplify()
```

(continues on next page)

(continued from previous page)

```
>>> psi.coefficient(sp.Rational(1, 2))
>>> psi.print()
|ψ⟩⟨ψ| = a*conjugate(a)|0⟩⟨0| + a*conjugate(b)|0⟩⟨1| + b*conjugate(a)|1⟩⟨0| +_
+ b*conjugate(b)|1⟩⟨1|
```

```
>>> from qhronology.mechanics.matrices import ket
>>> psi = QuantumState(spec=[("a", [0]), ("b", [1])], form="vector", label="ψ")
>>> psi.print()
|ψ⟩ = a|0⟩ + b|1⟩
>>> psi.measure(operators=[ket(0), ket(1)], observable=False, statistics=True)
[a*conjugate(a), b*conjugate(b)]
>>> psi.measure(operators=[ket(0), ket(1)], observable=False, statistics=False)
>>> psi.notation = "ρ"
>>> psi.print()
ρ = a*conjugate(a)|0⟩⟨0| + b*conjugate(b)|1⟩⟨1|
```

`QuantumState.postselect(
 postselections: list[tuple[mat | arr | QuantumObject, int]],
)`

Perform postselection on the state against the operators(s) specified in `postselections`.

The postselections can be given in either vector or matrix form. For the former, the transformation of the vector state $|\Psi\rangle$ follows the standard rule

$$|\Psi'\rangle = \langle\phi|\Psi\rangle \quad (8.4)$$

where $|\phi\rangle$ is the postselection vector. In the case of a matrix form $\hat{\omega}$, the notion of postselection of a density matrix state $\hat{\rho}$ naturally generalizes to

$$\hat{\rho}' = \text{tr}_{\{i\}}[\hat{\omega}\hat{\rho}] \quad (8.5)$$

where $\{i\}$ is the set of indices corresponding to the subsystem(s) upon which the postselection is performed.

If multiple postselections are supplied, the state will be successively postselected in the order in which they are given. If a vector state is postselected against a matrix form, it will automatically be transformed into its matrix form as necessary.

Parameters

`postselections (list[tuple[mat | arr | QuantumObject, int]])` – A list of 2-tuples of vectors or matrix operators paired with the first (smallest) index of their postselection target systems.

Note:

Any classes given in `postselections` that are derived from the `QuantumObject` base class (such as `QuantumState` (page 83) and `QuantumGate` (page 105)) will have their `symbols` and `conditions` properties merged into the current `QuantumState` (page 83) instance.

Examples

```
>>> psi = QuantumState(spec=[("a", [0, 0]), ("b", [1, 1])], form="vector", label="Ψ")
>>> phi = QuantumState(spec=[("c", [0]), ("d", [1])], form="vector", label="φ")
>>> psi.print()
|Ψ⟩ = a|0,0⟩ + b|1,1⟩
```

(continues on next page)

(continued from previous page)

```
>>> phi.print()
|φ⟩ = c|0⟩ + d|1⟩
>>> psi.postselect([(phi, [0])])
>>> psi.label = "Ψ"
>>> psi.print()
|Ψ'⟩ = a*conjugate(c)|0⟩ + b*conjugate(d)|1⟩
```

```
>>> from qhrontology.mechanics.matrices import ket
>>> psi = QuantumState(spec=[("a", [0, 0]), ("b", [1, 1])], form="vector", label="Ψ")
>>> psi.print()
|Ψ⟩ = a|0,0⟩ + b|1,1⟩
>>> psi.label = "Ψ"
>>> psi.postselect([(ket(0), [0])])
>>> psi.print()
|Ψ'⟩ = a|0⟩
>>> psi.reset()
>>> psi.postselect([(ket(1), [0])])
>>> psi.print()
|Ψ'⟩ = b|1⟩
```

8.1.5 Quantities

All of these methods are inherited from *QuantitiesMixin* (page 187).

QuantumState.trace() → Number | generic | Basic | MatrixSymbol | MatrixElement
| Symbol

Calculate the (complete) trace $\text{tr}[\hat{\rho}]$ of the internal state ($\hat{\rho}$).

Returns

num / sym – The trace of the internal state.

Examples

```
>>> state = QuantumState(
...     spec=[("a", [0]), ("b", [1])],
...     form="vector",
...     symbols={"a": {"complex": True}, "b": {"complex": True}},
...     conditions=[("a*conjugate(a) + b*conjugate(b)", 1)],
...     norm=1,
... )
>>> state.trace()
1
```

```
>>> state = QuantumState(
...     spec=[(1, [0]), (1, [1])],
...     kind="mixed",
...     symbols={"d": {"real": True}},
...     norm="1/d",
... )
>>> state.trace()
1/d
```

QuantumState.purity() → Number | generic | Basic | MatrixSymbol | MatrixElement
| Symbol

Calculate the purity (γ) of the internal state ($\hat{\rho}$):

$$\gamma(\hat{\rho}) = \text{tr}[\hat{\rho}^2]. \quad (8.6)$$

Returns

num / sym – The purity of the internal state.

Examples

```
>>> state = QuantumState(
...     spec=[("a", [0]), ("b", [1])],
...     form="vector",
...     symbols={"a": {"complex": True}, "b": {"complex": True}},
...     conditions=[("a*conjugate(a) + b*conjugate(b)", 1)],
...     norm=1,
... )
>>> state.purity()
1
```

```
>>> state = QuantumState(spec=[("p", [0]), ("1 - p", [1])], kind="mixed", norm=1)
>>> state.purity()
p**2 + (1 - p)**2
```

QuantumState.distance()

state: mat | QuantumObject,
→ num | sym

Calculate the trace distance (D) between the internal state ($\hat{\rho}$) and the given **state** ($\hat{\tau}$):

$$D(\hat{\rho}, \hat{\tau}) = \frac{1}{2}\text{tr}|\hat{\rho} - \hat{\tau}|. \quad (8.7)$$

Parameters

state (mat | QuantumObject) – The given state.

Returns

num / sym – The trace distance between the internal state and **state**.

Examples

```
>>> state_A = QuantumState(
...     spec=[("a", [0]), ("b", [1])],
...     form="vector",
...     symbols={"a": {"complex": True}, "b": {"complex": True}},
...     conditions=[("a*conjugate(a) + b*conjugate(b)", 1)],
...     norm=1,
... )
>>> state_B = QuantumState(
...     spec=[("c", [0]), ("d", [1])],
...     form="vector",
...     symbols={"c": {"complex": True}, "d": {"complex": True}},
...     conditions=[("c*conjugate(c) + d*conjugate(d)", 1)],
...     norm=1,
... )
>>> state_A.distance(state_A)
```

(continues on next page)

(continued from previous page)

```

0
>>> state_B.distance(state_B)
0
>>> state_A.distance(state_B)
sqrt((a*conjugate(b) - c*conjugate(d))*(b*conjugate(a) - d*conjugate(c)) +_
<-(b*conjugate(b) - d*conjugate(d))**2)/2 + sqrt((a*conjugate(a) -_
<-c*conjugate(c))**2 + (a*conjugate(b) - c*conjugate(d))*(b*conjugate(a) -_
<-d*conjugate(c)))/2

```

```

>>> state_A = QuantumState(
...     spec=[("p", [0]), ("1 - p", [1])],
...     kind="mixed",
...     symbols={"p": {"positive": True}},
...     norm=1,
... )
>>> state_B = QuantumState(
...     spec=[("q", [0]), ("1 - q", [1])],
...     kind="mixed",
...     symbols={"q": {"positive": True}},
...     norm=1,
... )
>>> state_A.distance(state_B)
Abs(p - q)

```

```

>>> plus_state = QuantumState(spec=[(1, [0]), (1, [1])], form="vector", norm=1)
>>> minus_state = QuantumState(spec=[(1, [0]), (-1, [1])], form="vector", norm=1)
>>> plus_state.distance(minus_state)
1

```

QuantumState.fidelity

```

state: mat | QuantumObject,
) → num | sym

```

Calculate the fidelity (F) between the internal state ($\hat{\rho}$) and the given `state` ($\hat{\tau}$):

$$F(\hat{\rho}, \hat{\tau}) = \left(\text{tr} \sqrt{\sqrt{\hat{\rho}} \hat{\tau} \sqrt{\hat{\rho}}} \right)^2. \quad (8.8)$$

Parameters

`state` (`mat` | `QuantumObject`) – The given state.

Returns

`num` / `sym` – The fidelity between the internal state and `state`.

Examples

```

>>> state_A = QuantumState(
...     spec=[("a", [0]), ("b", [1])],
...     form="vector",
...     symbols={"a": {"complex": True}, "b": {"complex": True}},
...     conditions=[("a*conjugate(a) + b*conjugate(b)", 1)],
...     norm=1,
... )
>>> state_B = QuantumState(
...     spec=[("c", [0]), ("d", [1])],
...

```

(continues on next page)

(continued from previous page)

```

...      form="vector",
...      symbols={"c": {"complex": True}, "d": {"complex": True}},
...      conditions=[("c*conjugate(c) + d*conjugate(d)", 1)],
...      norm=1,
... )
>>> state_A.fidelity(state_A)
1
>>> state_B.fidelity(state_B)
1
>>> state_A.fidelity(state_B)
(a*conjugate(c) + b*conjugate(d))*(c*conjugate(a) + d*conjugate(b))

```

```

>>> state_A = QuantumState(
...      spec=[("p", [0]), ("1 - p", [1])],
...      kind="mixed",
...      symbols={"p": {"positive": True}},
...      norm=1,
... )
>>> state_B = QuantumState(
...      spec=[("q", [0]), ("1 - q", [1])],
...      kind="mixed",
...      symbols={"q": {"positive": True}},
...      norm=1,
... )
>>> state_A.fidelity(state_B)
(sqrt(p)*sqrt(q) + sqrt((1 - p)*(1 - q)))*2

```

```

>>> plus_state = QuantumState(spec=[(1, [0]), (1, [1])], form="vector", norm=1)
>>> minus_state = QuantumState(spec=[(1, [0]), (-1, [1])], form="vector", norm=1)
>>> plus_state.fidelity(minus_state)
0

```

QuantumState.entropy

```

state: mat | QuantumObject = None,
base: num | None = None,
) → num | sym

```

Calculate the relative von Neumann entropy (S) between the internal state ($\hat{\rho}$) and the given `state` ($\hat{\tau}$):

$$S(\hat{\rho} \parallel \hat{\tau}) = \text{tr}[\hat{\rho}(\log_b \hat{\rho} - \log_b \hat{\tau})]. \quad (8.9)$$

If `state` is not specified (i.e., `None`), calculate the ordinary von Neumann entropy of the internal state ($\hat{\rho}$) instead:

$$S(\hat{\rho}) = \text{tr}[\hat{\rho} \log_b \hat{\rho}]. \quad (8.10)$$

Here, b represents `base`, which is the dimensionality of the unit of information with which the entropy is measured.

Parameters

- ▶ `state` (`mat` | `QuantumObject`) – The given state.
- ▶ `base` (`num`) – The dimensionality of the unit of information with which the entropy is measured. Defaults to 2.

Returns

`num` / `sym` – The (relative) von Neumann entropy.

Examples

```
>>> state_A = QuantumState(
...     spec=[("a", [0]), ("b", [1])],
...     form="vector",
...     symbols={"a": {"complex": True}, "b": {"complex": True}},
...     conditions=[("a*conjugate(a) + b*conjugate(b)", 1)],
...     norm=1,
... )
>>> state_B = QuantumState(
...     spec=[("c", [0]), ("d", [1])],
...     form="vector",
...     symbols={"c": {"complex": True}, "d": {"complex": True}},
...     conditions=[("c*conjugate(c) + d*conjugate(d)", 1)],
...     norm=1,
... )
>>> state_A.entropy()
0
>>> state_B.entropy()
0
>>> state_A.entropy(state_B)
0
```

```
>>> state_A = QuantumState(
...     spec=[("p", [0]), ("1 - p", [1])],
...     kind="mixed",
...     symbols={"p": {"positive": True}},
...     norm=1,
... )
>>> state_B = QuantumState(
...     spec=[("q", [0]), ("1 - q", [1])],
...     kind="mixed",
...     symbols={"q": {"positive": True}},
...     norm=1,
... )
>>> state_A.entropy()
(-p*log(p) + (p - 1)*log(1 - p))/log(2)
>>> state_B.entropy()
(-q*log(q) + (q - 1)*log(1 - q))/log(2)
>>> state_A.entropy(state_B)
(-(p - 1)*(log(1 - p) - log(1 - q)) + log((p/q)**p))/log(2)
```

`QuantumState.mutual`

`systems_A: int | list[int],
systems_B: int | list[int] | None = None,`
`) → Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol`

Calculate the mutual information (I) between two subsystems `systems_A` (A) and `systems_B` (B) of the internal state ($\rho^{A,B}$):

$$I(A : B) = S(\hat{\rho}^A) + S(\hat{\rho}^B) - S(\hat{\rho}^{A,B}) \quad (8.11)$$

where $S(\hat{\rho})$ is the von Neumann entropy of a state $\hat{\rho}$.

Parameters

- ▶ `systems_A (int | list[int])` – The indices of the first subsystem. Defaults to `[0]`.
- ▶ `systems_B (int | list[int])` – The indices of the second subsystem. Defaults to the complement of `systems_A` with respect to the entire composition of subsystems of `state`.

- **dim** (*int*) – The dimensionality of the composite quantum system (and its subsystems). Must be a non-negative integer. Defaults to **2**.

Returns

num / sym – The mutual information between the subsystems **systems_A** and **systems_B** of the internal state.

Examples

```
>>> state_AB = QuantumState(
...     spec=[("a", [0, 0]), ("b", [1, 1])],
...     form="vector",
...     symbols={"a": {"complex": True}, "b": {"complex": True}},
...     conditions=[("a*conjugate(a) + b*conjugate(b)", 1)],
...     norm=1,
... )
>>> state_AB.mutual([0], [1])
2*(-a*log(a*conjugate(a))*conjugate(a) - b*log(b*conjugate(b))*conjugate(b))/log(2)
```

```
>>> state_AB = QuantumState(
...     spec=[("a", [0, 0]), ("b", [1, 1])],
...     kind="mixed",
...     symbols={"a": {"positive": True}, "b": {"positive": True}},
...     conditions=[("a + b", 1)],
...     norm=1,
... )
>>> state_AB.mutual([0], [1])
(-a*log(a) - b*log(b))/log(2)
```

8.2 Subclasses

```
class VectorState(
    *args,
    **kwargs,
)
```

A specialized subclass for creating *vector* states and storing their metadata.

This is a wrapper on the **QuantumState** (page 83) class, and so inherits all of its attributes, properties, and methods. The distinction is that this **VectorState** class fixes both the **form** and **kind** arguments to the values of **"vector"** and **"pure"**, respectively, at instantiation. This means that neither ***args** or ****kwargs** must contain these arguments.

Examples

```
>>> qubit_vector = VectorState(spec=[(1, [0]), (1, [1])], norm=1)
>>> qubit_vector.print()
|ψ⟩ = sqrt(2)/2|0⟩ + sqrt(2)/2|1⟩
```

```
class MatrixState(
    *args,
    **kwargs,
)
```

A specialized subclass for creating *matrix* states and storing their metadata.

This is a wrapper on the `QuantumState` (page 83) class, and so inherits all of its attributes, properties, and methods. The distinction is that this `MatrixState` class fixes the `form` argument to a value of `"matrix"` at instantiation. This means that neither `*args` or `**kwargs` must contain this argument.

Examples

```
>>> qubit_matrix_pure = MatrixState(spec=[(1, [0]), (1, [1])], kind="pure", norm=1)
>>> qubit_matrix_pure.print()
|ψ⟩⟨ψ| = 1/2|0⟩⟨0| + 1/2|0⟩⟨1| + 1/2|1⟩⟨0| + 1/2|1⟩⟨1|
```

```
>>> qubit_matrix_mixed = MatrixState(spec=[(1, [0]), (1, [1])], kind="mixed", __
    ←norm=1)
>>> qubit_matrix_mixed.print()
ρ = 1/2|0⟩⟨0| + 1/2|1⟩⟨1|
```

```
class PureState(
    *args,
    **kwargs,
)
```

A specialized subclass for creating *pure* states and storing their metadata.

This is a wrapper on the `QuantumState` (page 83) class, and so inherits all of its attributes, properties, and methods. The distinction is that this `PureState` class fixes the `kind` argument to a value of `"pure"` at instantiation. This means that neither `*args` or `**kwargs` must contain this argument.

Examples

```
>>> qubit_pure_vector = PureState(spec=[(1, [0]), (1, [1])], form="vector", norm=1)
>>> qubit_pure_vector.print()
|ψ⟩ = sqrt(2)/2|0⟩ + sqrt(2)/2|1⟩
```

```
>>> qubit_pure_matrix = PureState(spec=[(1, [0]), (1, [1])], form="matrix", norm=1)
>>> qubit_pure_matrix.print()
|ψ⟩⟨ψ| = 1/2|0⟩⟨0| + 1/2|0⟩⟨1| + 1/2|1⟩⟨0| + 1/2|1⟩⟨1|
```

```
class MixedState(
    *args,
    **kwargs,
)
```

A specialized subclass for creating *mixed* states and storing their metadata.

This is a wrapper on the `QuantumState` (page 83) class, and so inherits all of its attributes, properties, and methods. The distinction is that this `MixedState` class fixes both the `form` and `kind` arguments to the values of `"matrix"` and `"mixed"`, respectively, at instantiation. This means that neither `*args` or `**kwargs` must contain these arguments.

Examples

```
>>> qubit_mixed = MixedState(spec=[(1, [0]), (1, [1])], norm=1)
>>> qubit_mixed.print()
ρ = 1/2|0⟩⟨0| + 1/2|1⟩⟨1|
```

Quantum logic gates provide the building blocks for describing quantum operations that are usually (unitary) interactions between two or more (sub)systems, or (linear) transformations of any number of systems. In Qchronology, they are constructed as instances of the [QuantumGate](#) (page 105) base class,

```
from qchronology.quantum.gates import QuantumGate
```

and its derivatives (subclasses),

```
from qchronology.quantum.gates import Pauli, GellMann, Rotation, Phase, Diagonal, Swap,
    Summation, Not, Hadamard, Fourier, Measurement
```

These represent a distinct vertical “slice” in the quantum circuit picturalism, and so include information about the locations of both control and anticontrol nodes, in addition to the presence of any empty wires. They also possess other metadata associated with the gate such as any parameter values, symbolic assumptions, and algebraic conditions.

Facilities to combine gates together are also provided by the package and take two forms: “interleaved” compositions via the [GateInterleave](#) (page 133) class, and “stacked” compositions via the [GateStack](#) (page 135) class:

```
from qchronology.quantum.gates import GateInterleave, GateStack
```

Both of these classes concern the creation of more complex spatial (“vertical”) gate structures. Temporal (“horizontal”) compositions (i.e., gate sequences) as single object instances on the other hand are not supported as this is achievable simply by combining the individual components sequentially in a circuit.

9.1 Main class

```
class QuantumGate(
    spec: MutableDenseMatrix | ndarray | list[list[Number | generic | Basic |
        MatrixSymbol | MatrixElement | Symbol | str]] | None,
    targets: list[int] | None = None,
    controls: list[int] | None = None,
    anticontrols: list[int] | None = None,
    num_systems: int | None = None,
    dim: int | None = None,
    symbols: dict | None = None,
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement |
        Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement |
        Symbol | str]] | None = None,
    conjugate: bool | None = None,
    exponent: Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol |
        str | None = None,
    coefficient: Number | generic | Basic | MatrixSymbol | MatrixElement | Sym-
        bol | str | None = None,
    label: str | None = None,
    notation: str | None = None,
    family: str | None = None,
)
Bases: QuantumObject
```

A class for creating quantum gates and storing their metadata.

This class forms the base upon which all quantum gates are built. Instances of this base class and its derivatives (subclasses) provide complete descriptions of quantum gates. This means that they describe a complete vertical column (or “slice”) in the quantum circuitry pictorialism, including control nodes, anticontrol nodes, empty wires, and the (unitary) gate operator itself. The details of any algebraic symbols, mathematical conditions, and visualization labels are also recorded. Note that, unlike the internal matrix representations contained within instances of the [QuantumState](#) (page 83) class (and its derivatives), the matrix representations of subclass instances of [QuantumGate](#) (page 105) are *not* mutable.

Parameters

- ▶ **spec** (`mat` | `arr` | `list[list[num | sym | str]]`) – The specification of the quantum gate’s matrix representation in a standard `dim`-dimensional basis. Can be one of:
 - a SymPy matrix (`mat`)
 - a NumPy array (`arr`)
 - a list of lists of numerical, symbolic, or string expressions that collectively specify a matrix (`list[list[num | sym | str]]`)
- Defaults to the single-system `dim`-dimensional Identity operator.
- ▶ **targets** (`list[int]`) – The numerical indices of the subsystems on which the gate elements reside. Defaults to `[0]` (if `num_systems` is `None`) or `[i for i in range(num_systems)]` (if `num_systems` is not `None`).
- ▶ **controls** (`list[int]`) – The numerical indices of the subsystems on which control nodes reside. Defaults to `[]`.
- ▶ **anticontrols** (`list[int]`) – The numerical indices of the subsystems on which anticontrol nodes reside. Defaults to `[]`.
- ▶ **num_systems** (`int`) – The (total) number of systems which the gate spans. Must be a non-negative integer. Defaults to `max(targets + controls + anticontrols + [count_systems(sp.Matrix(spec), dim)]) + 1`.
- ▶ **dim** (`int`) – The dimensionality of the quantum gate’s Hilbert space. Must be a non-negative integer. Defaults to `2`.
- ▶ **symbols** (`dict[sym | str, dict[str, Any]]`) – A dictionary in which the keys are individual symbols (usually found within the gate specification `spec`) and the values are dictionaries of their respective SymPy keyword-argument `assumptions`. Defaults to `{}`.
- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – A list of 2-tuples of conditions to be applied to the gate. All instances of the expression in each tuple’s first element are replaced by the expression in the respective second element. This uses the same format as the SymPy `subs()` method. The order in which they are applied is simply their order in the list. Defaults to `[]`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the gate when it is called. Defaults to `False`.
- ▶ **exponent** (`num | sym | str`) – A numerical or string representation of a scalar value to which gate’s operator (residing on `targets`) is exponentiated. Must be a non-negative integer. Useful for computing powers of gates (such as PSWAP), but is only guaranteed to return a valid power of a gate if its corresponding matrix representation (e.g., \hat{A}) is involutory (i.e., $\hat{A}^2 = \hat{I}$). Defaults to `1`.
- ▶ **coefficient** (`num | sym | str`) – A numerical or string representation of a scalar value by which the gate’s matrix (occupying `targets`) is multiplied. Performed after exponentiation. Useful for multiplying the gate by a phase factor. Defaults to `1`.
- ▶ **label** (`str`) – The unformatted string used to represent the gate in mathematical expressions. Defaults to `"U"`.

- ▶ **notation (str)** – The formatted string used to represent the gate in mathematical expressions. When not `None`, overrides the value passed to `label`. Not intended to be set by the user in most cases. Defaults to `None`.
- ▶ **family (str)** – A string expressing the kind of block element for which the gate is to be visualized. Not intended to be set by the user. Defaults to "`GATE`".

 **Note:**

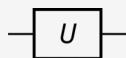
The indices specified in `targets`, `controls`, and `anticontrols` must be distinct.

Examples

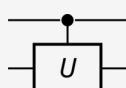
```
>>> unitary = sp.MatrixSymbol("U", 2, 2).asMutable()
>>> U = QuantumGate(spec=unitary, label="U")
>>> U.output()
Matrix([
[U[0, 0], U[0, 1]],
[U[1, 0], U[1, 1]])
```



```
>>> unitary = sp.MatrixSymbol("U", 3, 3).asMutable()
>>> U3 = QuantumGate(spec=unitary, label="U", dim=3)
>>> U3.output()
Matrix([
[U[0, 0], U[0, 1], U[0, 2]],
[U[1, 0], U[1, 1], U[1, 2]],
[U[2, 0], U[2, 1], U[2, 2]])
```



```
>>> unitary = sp.MatrixSymbol("U", 2, 2).asMutable()
>>> CU = QuantumGate(spec=unitary, targets=[1], controls=[0], label="U")
>>> CU.output()
Matrix([
[1, 0, 0, 0],
[0, 1, 0, 0],
[0, 0, U[0, 0], U[0, 1]],
[0, 0, U[1, 0], U[1, 1]])
```



9.1.1 Constructor argument properties

property `QuantumGate.spec: MutableDenseMatrix | ndarray | list[list[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]]`

The matrix representation of the quantum gate's operator. Provides a complete description of the operator in a standard `dim`-dimensional basis.

property `QuantumGate.targets: list[int]`

The numerical indices of the subsystems on which the gate elements reside.

property `QuantumGate.controls: list[int]`

The numerical indices of the subsystems on which control nodes reside.

For example, a controlled- \hat{U} gate in d dimensions takes the form

$$\begin{aligned} C^0 \hat{U}^1 &= \sum_{k=0}^{d-1} |k\rangle\langle k| \otimes \hat{U}^k \\ &= |0\rangle\langle 0| \otimes \hat{I} + |1\rangle\langle 1| \otimes \hat{U} + |2\rangle\langle 2| \otimes \hat{U}^2 + \dots + |d-1\rangle\langle d-1| \otimes \hat{U}^{d-1} \end{aligned} \tag{9.1}$$

property `QuantumGate.anticontrols: list[int]`

The numerical indices of the subsystems on which anticontrol nodes reside.

For example, an anticontrolled- \hat{U} gate in d dimensions takes the form

$$\begin{aligned} \bar{C}^0 \hat{U}^1 &= \sum_{k=0}^{d-1} |k\rangle\langle k| \otimes \hat{U}^{d-1-k} \\ &= |0\rangle\langle 0| \otimes \hat{U}^{d-1} + |1\rangle\langle 1| \otimes \hat{U}^{d-2} + |2\rangle\langle 2| \otimes \hat{U}^{d-3} + \dots + |d-1\rangle\langle d-1| \otimes \hat{I} \end{aligned} \tag{9.2}$$

property `QuantumGate.num_systems: int`

The number of systems that the object spans. Must be a non-negative integer. Should not be set for states.

property `QuantumGate.dim: int`

The dimensionality of the quantum object. Must be a non-negative integer.

property `QuantumGate.symbols: dict[MatrixSymbol | MatrixElement | Symbol | str, dict[str, Any]]`

A dictionary in which the keys are individual symbols (contained within the object's matrix representation) and the values are dictionaries of their respective SymPy keyword-argument `assumptions` ("predicates"). A full list of currently supported predicates, and their defaults, is as follows:

- ▶ "algebraic": True
- ▶ "commutative": True
- ▶ "complex": True
- ▶ "extended_negative": False
- ▶ "extended_nonnegative": True

- ▶ "extended_nonpositive": False
- ▶ "extended_nonzero": True
- ▶ "extended_positive": True
- ▶ "extended_real": True
- ▶ "finite": True
- ▶ "hermitian": True
- ▶ "imaginary": False
- ▶ "infinite": False
- ▶ "integer": True
- ▶ "irrational": False
- ▶ "negative": False
- ▶ "noninteger": False
- ▶ "nonnegative": True
- ▶ "nonpositive": False
- ▶ "nonzero": True
- ▶ "positive": True
- ▶ "rational": True
- ▶ "real": True
- ▶ "transcendental": False
- ▶ "zero": False

property QuantumGate.conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]]

A list of 2-tuples of conditions to be applied to the object's matrix representation.

property QuantumGate.conjugate: bool

Whether to perform Hermitian conjugation on the object when it is called.

property QuantumGate.exponent: Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str

A numerical or string representation of a scalar value specifying the value to which the gate's matrix representation is exponentiated. Is guaranteed to produce valid powers only for involutory matrices.

For an involutory matrix \hat{A} , that is $\hat{A}^2 = \hat{I}$ (where \hat{I} is the identity matrix), we have the identity,

$$\exp[ix\hat{A}] = \cos(x)\hat{I} + i \sin(x)\hat{A}, \quad (9.3)$$

for any $x \in \mathbb{C}$. In the case of $x = -\frac{\pi}{2}$, this becomes

$$\exp\left[-i\frac{\pi}{2}\hat{A}\right] = -i\hat{A}, \quad (9.4)$$

which can be rearranged to give

$$\begin{aligned}\hat{A} &= i \exp\left[-i \frac{\pi}{2} \hat{A}\right] \\ &= \exp\left[i \frac{\pi}{2}\right] \cdot \exp\left[-i \frac{\pi}{2} \hat{A}\right].\end{aligned}\tag{9.5}$$

Simply taking this expression to an arbitrary power $p \in \mathbb{C}$ thus yields the identity

$$\begin{aligned}\hat{A}^p &= \exp\left[i \frac{\pi}{2} p\right] \cdot \exp\left[-i \frac{\pi}{2} p \hat{A}\right] \\ &= \exp\left[i \frac{\pi}{2} p\right] \left[\cos\left(\frac{\pi}{2} p\right) \hat{I} - i \sin\left(\frac{\pi}{2} p\right) \hat{A} \right] \\ &= \frac{1 + e^{i\pi p}}{2} \hat{I} + \frac{1 - e^{i\pi p}}{2} \hat{A}.\end{aligned}\tag{9.6}$$

property QuantumGate.coefficient: Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str

A numerical or string representation of a scalar value by which the gate's matrix (occupying **targets**) is multiplied.

property QuantumGate.label: str

The unformatted string used to represent the object in mathematical expressions. Must have a non-zero length.

property QuantumGate.notation: str

The formatted string used to represent the object in mathematical expressions. When set, overrides the value of the **label** property. Must have a non-zero length. Not intended to be set by the user in most cases.

property QuantumGate.family: str

The code of the block element that the object is to be visualized as. Not intended to be set by the user.

9.1.2 Read-only properties

property QuantumGate.systems: list[int]

Read-only property containing an ordered list of the numerical indices of the object's systems.

property QuantumGate.matrix: MutableDenseMatrix

The matrix representation of the total gate across all of its systems.

9.1.3 Methods

```
QuantumGate.output()
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
    simplify: bool | None = None,
    conjugate: bool | None = None,
```

```

exponent: bool | Number | generic | Basic | MatrixSymbol | MatrixElement |
Symbol | str | None = None,
coefficient: bool | Number | generic | Basic | MatrixSymbol | MatrixElement
| Symbol | str | None = None,
) → MutableDenseMatrix

```

Construct the gate and return its matrix representation.

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the gate. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the gate. Defaults to `False`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the gate. If `False`, does not conjugate. Defaults to the value of `self.conjugate`.
- ▶ **exponent** (`bool | num | sym | str`) – The scalar value by which the gate's matrix representation is exponentiated. If `False`, does not exponentiate. Defaults to the value of `self.exponent`.
- ▶ **coefficient** (`num | sym | str`) – The scalar value by which the gate's matrix representation is multiplied. If `False`, does not multiply the gate by the coefficient. Defaults to the value of `self.coefficient`.

Returns

`mat` – The constructed quantum gate.

```

QuantumGate.diagram(
    pad: tuple[int, int] | None = None,
    sep: tuple[int, int] | None = None,
    style: str | None = None,
    return_string: bool | None = None,
) → None | str

```

Print or return a string diagram of the quantum object as a multiline string.

Parameters

- ▶ **pad** (`tuple[int, int]`) – A two-tuple describing the horizontal and vertical interior paddings between the content at the centre of the object (e.g., its label) and its outer edge (e.g., block border). Both integers must be non-negative. Defaults to `(0, 0)`.
- ▶ **sep** (`tuple[int, int]`) – A two-tuple describing the horizontal and vertical exterior separation distances at the object's edges. Both integers must be non-negative. Defaults to `(1, 1)`.
- ▶ **style** (`str`) – A string specifying the style for the circuit visualization to take. Can be any of `"ascii"`, `"unicode"`, or `"unicode_alt"`. Defaults to `"unicode"`.
- ▶ **return_string** (`bool`) – Whether to return the assembled diagram as a multiline string. Defaults to `False`.

Returns

- ▶ `None` – Returned only if `return_string` is `False`.
- ▶ `str` – The rendered circuit diagram. Returned only if `return_string` is `True`.

Note:

The quality of the visualization depends greatly on the output's configuration. For best results, the terminal should have a monospace font with good Unicode coverage.

Examples

For usage examples, please see those of the [QuantumGate](#) (page 105) class and its *subclasses* (page 112).

9.2 Subclasses

Please note that the documentation of these subclasses includes only properties and methods that are either new or modified from the base class [QuantumGate](#) (page 105).

i Note:

In all of these subclasses, the `spec` property should not be set.

Table 9.1: List of aliases for the [QuantumGate](#) subclasses

Subclass	Alias
Pauli (page 112)	PAULI
GellMann (page 115)	GM
Rotation (page 116)	ROT
Phase (page 117)	PHS
Diagonal (page 119)	DIAG
Swap (page 121)	SWAP
Summation (page 124)	SUM
Not (page 125)	NOT
Hadamard (page 127)	HAD
Fourier (page 129)	QDFT
Measurement (page 131)	METER

```
class Pauli(  
    *args,  
    index: int,  
    **kwargs,  
)  
Bases: QuantumGate (page 105)
```

A subclass for creating Pauli gates and storing their metadata.

This is built upon the [QuantumGate](#) (page 105) class, and so inherits all of its attributes, properties, and methods.

This is fundamentally a single-system gate, and so a copy is placed on each of the subsystems corresponding to the indices in the `targets` property.

Parameters

- ▶ `*args` – Variable-length argument list, passed directly to the constructor `__init__` of the superclass [QuantumGate](#) (page 105).
- ▶ `index (int)` – The index of the desired Pauli matrix. Can take the following values:
 - **0** (2-dimensional identity matrix \hat{I})
 - **1** (Pauli-X $\hat{\sigma}_x$)
 - **2** (Pauli-Y $\hat{\sigma}_y$)
 - **3** (Pauli-Z $\hat{\sigma}_z$)

- ▶ ****kwargs** – Arbitrary keyword arguments, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).

 **Note:**

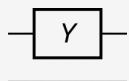
The Pauli gates are defined only for 2-dimensional (i.e., binary/qubit) systems. This means that the constructor does not take `dim` as an argument, nor can the associated property be set.

Examples

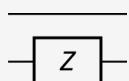
```
>>> X = Pauli(index=1)
>>> X.output()
Matrix([
[0, 1],
[1, 0]])
>>> X.diagram()
```



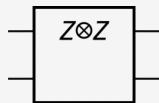
```
>>> YI = Pauli(index=2, targets=[0], num_systems=2)
>>> YI.output()
Matrix([
[0, 0, -I, 0],
[0, 0, 0, -I],
[I, 0, 0, 0],
[0, I, 0, 0]])
>>> YI.diagram()
```



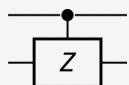
```
>>> IZ = Pauli(index=3, targets=[1])
>>> IZ.output()
Matrix([
[1, 0, 0, 0],
[0, -1, 0, 0],
[0, 0, 1, 0],
[0, 0, 0, -1]])
>>> IZ.diagram()
```



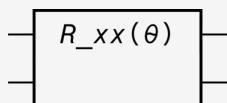
```
>>> ZZ = Pauli(index=3, targets=[0, 1], label="Z⊗Z")
>>> ZZ.output()
Matrix([
[1, 0, 0, 0],
[0, -1, 0, 0],
[0, 0, -1, 0],
[0, 0, 0, 1]])
>>> ZZ.diagram()
```



```
>>> CZ = Pauli(index=3, targets=[1], controls=[0])
>>> CZ.output()
Matrix([
[1, 0, 0, 0],
[0, 1, 0, 0],
[0, 0, 1, 0],
[0, 0, 0, -1]])
>>> CZ.diagram()
```



```
>>> R_xx = Pauli(
...     index=1,
...     targets=[0, 1],
...     exponent="θ/pi",
...     coefficient="exp(-I*θ/2)",
...     label="R_xx(θ)",
... )
>>> R_xx.output(simplify = True)
Matrix([
[cos(θ/2), 0, 0, -I*sin(θ/2)],
[0, cos(θ/2), -I*sin(θ/2), 0],
[0, -I*sin(θ/2), cos(θ/2), 0],
[-I*sin(θ/2), 0, 0, cos(θ/2)]])
>>> R_xx.diagram()
```



property `index: int`

The index of the desired Pauli matrix.

```
class GellMann(
    *args,
    index: int,
    **kwargs,
)
```

Bases: `QuantumGate` (page 105)

A subclass for creating Gell-Mann gates and storing their metadata.

This is built upon the `QuantumGate` (page 105) class, and so inherits all of its attributes, properties, and methods.

This is fundamentally a single-system gate, and so a copy is placed on each of the subsystems corresponding to the indices in the `targets` property.

Parameters

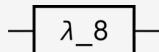
- ▶ `*args` – Variable-length argument list, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).
- ▶ `index (int)` – The index of the desired Gell-Mann matrix. Can take the following values:
 - `0` (3-dimensional identity matrix \hat{I})
 - `1` ($\hat{\lambda}_1$)
 - `2` ($\hat{\lambda}_2$)
 - `3` ($\hat{\lambda}_3$)
 - `4` ($\hat{\lambda}_4$)
 - `5` ($\hat{\lambda}_5$)
 - `6` ($\hat{\lambda}_6$)
 - `7` ($\hat{\lambda}_7$)
 - `8` ($\hat{\lambda}_8$)
- ▶ `**kwargs` – Arbitrary keyword arguments, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).

Note:

The Gell-Mann gates are defined only for 3-dimensional (i.e., ternary/qutrit) systems. This means that the constructor does not take `dim` as an argument, nor can the associated property be set.

Examples

```
>>> L = GellMann(index=8)
>>> L.output()
Matrix([
[sqrt(3)/3, 0, 0],
[0, sqrt(3)/3, 0],
[0, 0, -2*sqrt(3)/3]])
>>> L.diagram()
```


property index: int

The index of the desired Gell-Mann matrix.

```
class Rotation(
    *args,
    axis: int,
    angle: Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol |
    str | None = None,
    **kwargs,
)
```

Bases: [QuantumGate](#) (page 105)

A subclass for creating rotation gates and storing their metadata.

This is built upon the [QuantumGate](#) (page 105) class, and so inherits all of its attributes, properties, and methods.

The elementary rotation matrices \hat{R}_i are a set of three 2×2 matrices,

$$\begin{aligned}\hat{R}_x &= e^{-i\hat{\sigma}_x\theta/2} = \begin{bmatrix} \cos(\theta/2) & -i \sin(\theta/2) \\ -i \sin(\theta/2) & \cos(\theta/2) \end{bmatrix} \\ \hat{R}_y &= e^{-i\hat{\sigma}_y\theta/2} = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix} \\ \hat{R}_z &= e^{-i\hat{\sigma}_z\theta/2} = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}\end{aligned}\tag{9.7}$$

where θ is the rotation angle ([angle](#)).

These are fundamentally single-system gates, and so a copy of the specified gate is placed on each of the subsystems corresponding to the indices in the [targets](#) property.

Parameters

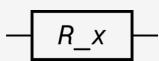
- ▶ ***args** – Variable-length argument list, passed directly to the constructor [__init__](#) of the superclass [QuantumGate](#) (page 105).
- ▶ **axis (int)** – The index corresponding to the axis of the desired rotation matrix. Can take the following values:
 - **1** (*x*-rotation \hat{R}_x)
 - **2** (*y*-rotation \hat{R}_y)
 - **3** (*z*-rotation \hat{R}_z)
- ▶ **angle (num | sym | str)** – The scalar value to be used as the rotation angle. Defaults to **0**.
- ▶ ****kwargs** – Arbitrary keyword arguments, passed directly to the constructor [__init__](#) of the superclass [QuantumGate](#) (page 105).

Note:

The rotation gates are defined only for 2-dimensional (i.e., binary/qubit) systems. This means that the constructor does not take [dim](#) as an argument, nor can the associated property be set.

Examples

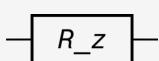
```
>>> R_x = Rotation(axis=1, angle="θ", label="R_x")
>>> R_x.output()
Matrix([
[cos(θ/2), -I*sin(θ/2)],
[-I*sin(θ/2), cos(θ/2)]])
>>> R_x.diagram()
```



```
>>> R_y = Rotation(axis=2, angle="φ", label="R_y")
>>> R_y.output()
Matrix([
[cos(φ/2), -sin(φ/2)],
[sin(φ/2), cos(φ/2)]])
>>> R_y.diagram()
```



```
>>> R_z = Rotation(axis=3, angle="t", label="R_z")
>>> R_z.output()
Matrix([
[exp(-I*t/2), 0],
[0, exp(I*t/2)]])
>>> R_z.diagram()
```



property axis: int

The index corresponding to the axis of the desired rotation matrix.

property angle: Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str

The scalar value to be used as the rotation angle.

```
class Phase(
    *args,
    phase: Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol |
    str | None = None,
    **kwargs,
)
```

Bases: [QuantumGate](#) (page 105)

A subclass for creating phase gates and storing their metadata.

This is built upon the [QuantumGate](#) (page 105) class, and so inherits all of its attributes, properties, and methods.

In d dimensions, a phase operator \hat{P} may be represented as a $d \times d$ diagonal matrix

$$\hat{P}(\omega) = \sum_{k=0}^{d-1} \omega^k |k\rangle\langle k| \quad (9.8)$$

where ω is the *phase* factor.

This is fundamentally a single-system gate, and so a copy is placed on each of the subsystems corresponding to the indices in the `targets` property.

Parameters

- ▶ `*args` – Variable-length argument list, passed directly to the constructor `__init__` of the superclass [QuantumGate](#) (page 105).
- ▶ `phase (num | sym | str)` – The phase value. Defaults to the unit root given by `sp.exp(2 * sp.pi * sp.I / self.dim)`.
- ▶ `**kwargs` – Arbitrary keyword arguments, passed directly to the constructor `__init__` of the superclass [QuantumGate](#) (page 105).

Examples

```
>>> P = Phase()
>>> P.output()
Matrix([
[1, 0],
[0, -1])
>>> P.diagram()
```



```
>>> S = Phase(exponent=sp.Rational(1, 2), label="S")
>>> S.output()
Matrix([
[1, 0],
[0, I])
>>> S.diagram()
```



```
>>> T = Phase(exponent=sp.Rational(1, 4), label="T")
>>> T.output()
Matrix([
[1, 0],
```

(continues on next page)

(continued from previous page)

```
[0, exp(I*pi/4)])
>>> D.diagram()
```



```
>>> P3 = Phase(dim=3)
>>> P3.output()
Matrix([
[1, 0, 0],
[0, exp(2*I*pi/3), 0],
[0, 0, exp(-2*I*pi/3)])
>>> P3.diagram()
```



```
>>> W = Phase(phase="w", dim=3, label="W")
>>> W.output()
Matrix([
[1, 0, 0],
[0, w, 0],
[0, 0, w**2])
>>> W.diagram()
```



property phase: Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str

The phase value.

```
class Diagonal(
    *args,
    entries: dict[int | list[int], Number | generic | Basic | MatrixSymbol |
    MatrixElement | Symbol | str],
    exponentiation: bool | None = None,
    **kwargs,
)
```

Bases: [QuantumGate](#) (page 105)

A subclass for creating diagonal gates and storing their metadata.

This is built upon the [QuantumGate](#) (page 105) class, and so inherits all of its attributes, properties, and methods.

In d dimensions, a diagonal operator \hat{D} may be represented as a $d \times d$ diagonal matrix

$$\hat{D}(\lambda_0, \lambda_1, \dots, \lambda_{d-1}) = \sum_{k=0}^{d-1} \lambda_k |k\rangle\langle k|, \quad \lambda_k \in \mathbb{C}, |\lambda_k| = 1 \quad (9.9)$$

where $\{\lambda_k\}_{k=0}^{d-1}$ are the main diagonal *entries*.

This is fundamentally a single-system gate, and so a copy is placed on each of the subsystems corresponding to the indices in the `targets` property.

Parameters

- ▶ `*args` – Variable-length argument list, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).
- ▶ `entries (dict[int | list[int], num | sym | str])` – A dictionary in which the keys are level specifications (integer or list of integers) and the values are scalars.
- ▶ `exponentiation (bool)` – Whether to exponentiate (with imaginary unit) the values given in `entries`. Defaults to `False`.
- ▶ `**kwargs` – Arbitrary keyword arguments, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).

 **Note:**

Levels that are unspecified in the `entries` argument all have a corresponding matrix element of `1`, regardless of the value of `exponentiation`.

Examples

```
>>> D = Diagonal(entries={0: "u", 1: "v"}, exponentiation=False)
>>> D.output()
Matrix([
[u, 0],
[0, v])
>>> D.diagram()
```



```
>>> P = Diagonal(
...     entries={1: "p"},
...     exponentiation=True,
...     symbols={"p": {"real": True}},
...     label="P",
... )
>>> P.output()
Matrix([
[1, 0],
[0, exp(I*p)])
>>> P.diagram()
```



```
>>> D3 = Diagonal(
...     entries={0: "a", 1: "b", 2: "c"},
...     exponentiation=False,
...     symbols={"a": {"real": True}, "b": {"real": True}, "c": {"real": True}},
...     dim=3,
... )
>>> D3.output()
Matrix([
[a, 0, 0],
[0, b, 0],
[0, 0, c])
>>> D3.diagram()
```



property `entries: dict[int | list[int], Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]`

A dictionary in which the keys are level specifications (integer or list of integers) and the values are scalars.

property `exponentiation: bool`

Whether to exponentiate (with imaginary unit) the values given in `entries`.

```
class Swap(
    *args,
    **kwargs,
)
```

Bases: `QuantumGate` (page 105)

A subclass for creating SWAP (exchange) gates and storing their metadata.

This is built upon the `QuantumGate` (page 105) class, and so inherits all of its attributes, properties, and methods.

In d dimensions, a SWAP operator \hat{S} between two (neighbouring) systems A and B may be represented as a $d^2 \times d^2$ matrix

$$\hat{S}^{A,B} = \sum_{j,k=0}^{d-1} |j\rangle\langle k|^A \otimes |k\rangle\langle j|^B, \quad (9.10)$$

where the identity operator acts on all other systems.

Parameters

- ▶ `*args` – Variable-length argument list, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).

- ▶ **targets** (*list[int, int]*) – A list of exactly two indices corresponding to the systems to be swapped. Is an argument of the superclass *QuantumGate* (page 105), so can be specified positionally in **args*.
- ▶ ****kwargs** – Arbitrary keyword arguments, passed directly to the constructor *__init__* of the superclass *QuantumGate* (page 105).

Examples

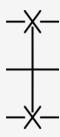
```
>>> S = Swap(targets=[0, 1])
>>> S.output()
Matrix([
[1, 0, 0, 0],
[0, 0, 1, 0],
[0, 1, 0, 0],
[0, 0, 0, 1]])
>>> S.diagram()
```



```
>>> S3 = Swap(targets=[0, 1], dim=3)
>>> S3.output()
Matrix([
[1, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1]])
>>> S3.diagram()
```



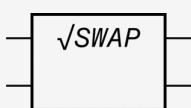
```
>>> SIS = Swap(targets=[0, 2])
>>> SIS.output()
Matrix([
[1, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1]])
>>> SIS.diagram()
```



```
>>> SCS = Swap(targets=[0, 2], controls=[1])
>>> SCS.output()
Matrix([
[1, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1]])
>>> SCS.diagram()
```



```
>>> RSWAP = Swap(targets=[0, 1], exponent=sp.Rational(1, 2), label="✓SWAP", family=
... "GATE")
>>> RSWAP.output()
Matrix([
[1, 0, 0, 0],
[0, 1/2 + I/2, 1/2 - I/2, 0],
[0, 1/2 - I/2, 1/2 + I/2, 0],
[0, 0, 0, 1]])
>>> RSWAP.diagram()
```



```
>>> PSWAP = Swap(
...     targets=[0, 1],
...     exponent="p",
...     symbols={"p": {"real": True}},
...     label="SWAP^p",
...     family="GATE",
... )
>>> PSWAP.output()
Matrix([
[1, 0, 0, 0],
[0, exp(I*pi*p)/2 + 1/2, 1/2 - exp(I*pi*p)/2, 0],
```

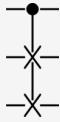
(continues on next page)

(continued from previous page)

```
[0, 1/2 - exp(I*pi*p)/2, exp(I*pi*p)/2 + 1/2, 0],
[0, 0, 0, 1])
>>> PSWAP.diagram()
```



```
>>> CSWAP = Swap(targets=[1, 2], controls=[0])
>>> CSWAP.output()
Matrix([
[1, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 1])
>>> CSWAP.diagram()
```



```
class Summation(
    *args,
    shift: int | None = None,
    **kwargs,
)
```

Bases: [QuantumGate](#) (page 105)

A subclass for creating SUM (summation) gates and storing their metadata.

This is built upon the [QuantumGate](#) (page 105) class, and so inherits all of its attributes, properties, and methods.

The SUM gate is essentially a generalization of the NOT gate. In d dimensions, it is defined as the operator

$$\hat{\Sigma}(n) = \sum_{k=0}^{d-1} |k \oplus n\rangle\langle k| \quad (9.11)$$

where n (`shift`) is the `shift` parameter, and $k \oplus n \equiv k + n \bmod d$.

The case of $n = 1$ is known as the `shift` operator, and represents a (non-Hermitian) generalization of the Pauli-X $\hat{\sigma}_x$ operator to d dimensions.

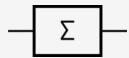
This is fundamentally a single-system gate, and so a copy is placed on each of the subsystems corresponding to the indices in the `targets` property.

Parameters

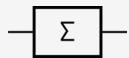
- ▶ ***args** – Variable-length argument list, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).
- ▶ **shift** (`int`) – The summation shift parameter. Must be a non-negative integer. Defaults to `1`.
- ▶ ****kwargs** – Arbitrary keyword arguments, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).

Examples

```
>>> SUM = Summation(shift=1)
>>> SUM.output()
Matrix([
[0, 1],
[1, 0]])
>>> SUM.diagram()
```



```
>>> SUM3 = Summation(shift=1, dim=3)
>>> SUM3.output()
Matrix([
[0, 0, 1],
[1, 0, 0],
[0, 1, 0]])
>>> SUM3.diagram()
```



`property shift: int`

The summation shift parameter.

```
class Not(
    *args,
    **kwargs,
)
```

Bases: `Summation` (page 124)

A subclass for creating NOT (negation or “bit-flip”) gates and storing their metadata.

This is built upon the `QuantumGate` (page 105) class, and so inherits all of its attributes, properties, and methods.

The NOT gate is essentially a specialization of the SUM gate to 2-dimensional (i.e., binary/qubit) systems, and is exactly equivalent to the Pauli-X gate. As such, this class exists purely to simplify access to this operation.

This is fundamentally a single-system gate, and so a copy is placed on each of the subsystems corresponding to the indices in the `targets` property.

Parameters

- ▶ ***args** – Variable-length argument list, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).
- ▶ ****kwargs** – Arbitrary keyword arguments, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).

Note:

NOT gates are defined only for 2-dimensional (i.e., binary/qubit) systems. This means that the constructor does not take `dim` as an argument, nor can the associated property be set.

Examples

```
>>> N = Not(targets=[0])
>>> N.output()
Matrix([
[0, 1],
[1, 0]])
>>> N.diagram()
```



```
>>> NN = Not(targets=[0, 1])
>>> NN.output()
Matrix([
[0, 0, 0, 1],
[0, 0, 1, 0],
[0, 1, 0, 0],
[1, 0, 0, 0]])
>>> NN.diagram()
```



```
>>> CNOT = Not(targets=[1], controls=[0])
>>> CNOT.output()
Matrix([
[1, 0, 0, 0],
[0, 1, 0, 0],
[0, 0, 0, 1],
[0, 0, 1, 0]])
>>> CNOT.diagram()
```



```
>>> ANOT = Not(targets=[1], anticontrols=[0])
>>> ANOT.output()
Matrix([
[0, 1, 0, 0],
[1, 0, 0, 0],
[0, 0, 1, 0],
[0, 0, 0, 1]])
>>> ANOT.diagram()
```



```
>>> CCNOT = Not(targets=[2], controls=[0, 1])
>>> CCNOT.output()
Matrix([
[1, 0, 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],
[0, 0, 0, 1, 0, 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, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0]])
>>> CCNOT.diagram()
```



```
>>> RNOT = Not(targets=[0], exponent=sp.Rational(1, 2))
>>> RNOT.output()
Matrix([
[1/2 + I/2, 1/2 - I/2],
[1/2 - I/2, 1/2 + I/2]])
>>> RNOT.diagram()
```



```
class Hadamard(
    *args,
    **kwargs,
)
```

Bases: [QuantumGate](#) (page 105)

A subclass for creating Hadamard gates and storing their metadata.

This is built upon the [QuantumGate](#) (page 105) class, and so inherits all of its attributes, properties, and methods.

The elementary Hadamard gate \hat{H} (for qubits) may be represented as the 2×2 operator

$$\begin{aligned}\hat{H} &= \frac{1}{\sqrt{2}} \sum_{j,k=0}^1 (-1)^{jk} |j\rangle\langle k| \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.\end{aligned}\tag{9.12}$$

This can be generalized to the following d -dimensional form for qudits,

$$\begin{aligned}\hat{H}_d &= \frac{1}{\sqrt{d}} \sum_{j,k=0}^{d-1} \omega_d^{k(d-j)} |j\rangle\langle k| \\ &= \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^{d-1} & \omega^{2(d-1)} & \dots & \omega^{(d-1)^2} \\ 1 & \omega^{d-2} & \omega^{2(d-2)} & \dots & \omega^{(d-1)(d-2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega & \omega^2 & \dots & \omega^{d-1} \end{bmatrix}\end{aligned}\tag{9.13}$$

where $\omega_d \equiv e^{\frac{2\pi i}{d}}$.

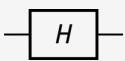
This is fundamentally a single-system gate, and so a copy is placed on each of the subsystems corresponding to the indices in the [targets](#) property.

Parameters

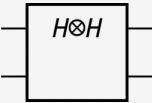
- ▶ ***args** – Variable-length argument list, passed directly to the constructor [`__init__`](#) of the superclass [QuantumGate](#) (page 105).
- ▶ ****kwargs** – Arbitrary keyword arguments, passed directly to the constructor [`__init__`](#) of the superclass [QuantumGate](#) (page 105).

Examples

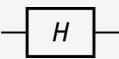
```
>>> H = Hadamard()
>>> H.output()
Matrix([
[sqrt(2)/2,  sqrt(2)/2],
[sqrt(2)/2, -sqrt(2)/2]])
```



```
>>> HH = Hadamard(targets=[0, 1], label="H⊗H")
>>> HH.output()
Matrix([
[1/2,  1/2,  1/2,  1/2],
[1/2, -1/2,  1/2, -1/2],
[1/2,  1/2, -1/2, -1/2],
[1/2, -1/2, -1/2,  1/2]])
```



```
>>> H3 = Hadamard(dim=3)
>>> H3.output()
Matrix([
[sqrt(3)/3, sqrt(3)/3, sqrt(3)/3],
[sqrt(3)/3, sqrt(3)*exp(-2*I*pi/3)/3, sqrt(3)*exp(2*I*pi/3)/3],
[sqrt(3)/3, sqrt(3)*exp(2*I*pi/3)/3, sqrt(3)*exp(-2*I*pi/3)/3]])
>>> H3.diagram()
```



```
class Fourier(
    *args,
    composite: bool | None = None,
    reverse: bool | None = None,
    **kwargs,
)
```

Bases: [QuantumGate](#) (page 105)

A subclass for creating Fourier (quantum discrete Fourier transform [QDFT]) gates and storing their metadata.

This is built upon the [QuantumGate](#) (page 105) class, and so inherits all of its attributes, properties, and methods.

The elementary Fourier operator \hat{F} for a single d -dimensional qudit may be represented as the $d \times d$ matrix

$$\begin{aligned} \hat{F} &= \frac{1}{\sqrt{d}} \sum_{j,k=0}^{d-1} \omega_d^{jk} |j\rangle\langle k| \\ &= \frac{1}{\sqrt{d}} \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \dots & \omega^{d-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \dots & \omega^{2(d-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \dots & \omega^{3(d-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{d-1} & \omega^{2(d-1)} & \omega^{3(d-1)} & \dots & \omega^{(d-1)(d-1)} \end{bmatrix} \end{aligned} \quad (9.14)$$

where $\omega_d = e^{\frac{2\pi i}{d}} = \omega$.

In the case of N qudits, the action of the multipartite Fourier operator \hat{F}_N on the basis state $\bigotimes_{\ell=1}^N |j_\ell\rangle \equiv |j_1, \dots, j_N\rangle$ (where $j_\ell \in \mathbb{Z}_0^{d-1}$) is

$$|j_1, \dots, j_N\rangle \xrightarrow{\hat{F}_N} \frac{1}{\sqrt{d^N}} \bigotimes_{\ell=1}^N \sum_{k_\ell=0}^{d-1} e^{2\pi i j_\ell k_\ell d^{-\ell}} |k_\ell\rangle \quad (9.15)$$

where $j \equiv \sum_{\ell=1}^N j_\ell d^{N-\ell}$.

If `composite` is `True`, a copy of the elementary form \hat{F} is placed on each of the subsystems corresponding to the indices in the `targets` property.

If `composite` is `False`, the composite form \hat{F}_N is applied to the subsystems specified by `targets` in:

- ▶ *ascending* order if `reverse` is `False`
- ▶ *descending* order if `reverse` is `True`

Parameters

- ▶ `*args` – Variable-length argument list, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).
- ▶ `composite (bool)` – Whether the composite (multipartite) Fourier gate is to be used. If `False`, copies of the elementary Fourier gate are placed on each index specified in `targets`. Defaults to `True`.
- ▶ `reverse (bool)` – Whether to reverse the order in which the composite (multipartite) Fourier gate is applied. Only applies when `composite` is `False`. Defaults to `False`.
- ▶ `**kwargs` – Arbitrary keyword arguments, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).

Examples

```
>>> F = Fourier()
>>> F.output()
Matrix([
[sqrt(2)/2,  sqrt(2)/2],
[sqrt(2)/2, -sqrt(2)/2]])
>>> F.diagram()
```



```
>>> F3 = Fourier(dim=3)
>>> F3.output()
Matrix([
[sqrt(3)/3,           sqrt(3)/3,           sqrt(3)/3],
[sqrt(3)/3, sqrt(3)*exp(-2*I*pi/3)/3, sqrt(3)*exp(2*I*pi/3)/3],
[sqrt(3)/3, sqrt(3)*exp(2*I*pi/3)/3, sqrt(3)*exp(-2*I*pi/3)/3])
>>> F3.diagram()
```

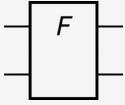


```
>>> FF = Fourier(targets=[0, 1])
>>> FF.output()
Matrix([
[1/2,  1/2,  1/2,  1/2],
[1/2, -1/2,  1/2, -1/2],
[1/2,  I/2, -1/2, -I/2],
```

(continues on next page)

(continued from previous page)

```
[1/2, -I/2, -1/2, I/2])
>>> FF.diagram()
```

**`property composite: bool`**

Whether the composite (multipartite) Fourier gate is to be used.

`property reverse: bool`

Whether to reverse the order in which the composite (multipartite) Fourier gate is applied.
Has no effect when `self.composite` is `False`.

```
class Measurement(
    *args,
    operators: list[MutableDenseMatrix | ndarray | QuantumObject],
    observable: bool | None = None,
    **kwargs,
)
```

Bases: `QuantumGate` (page 105)

A subclass for creating measurement gates and storing their metadata.

This is built upon the `QuantumGate` (page 105) class, and so inherits all of its attributes, properties, and methods.

Instances of this class each describe a (non-linear) operation in which the input state ($\hat{\rho}$) is quantum-mechanically *measured* (against the forms in specified in `operators`) and subsequently mutated according to its predicted post-measurement form (i.e., the sum of all possible measurement outcomes). This yields the transformed states:

- ▶ When `observable` is `False` (`operators` is a list of Kraus operators or projectors \hat{K}_i):

$$\hat{\rho}' = \sum_i \hat{K}_i \hat{\rho} \hat{K}_i^\dagger. \quad (9.16)$$

- ▶ When `observable` is `True` (`operators` is a list of observables \hat{O}_i):

$$\hat{\rho}' = \sum_i \text{tr}[\hat{O}_i \hat{\rho}] \hat{O}_i. \quad (9.17)$$

The items in the list `operators` can also be vectors (e.g., $|\xi_i\rangle$), in which case each is converted into its corresponding matrix representation (e.g., $\hat{K}_i = |\xi_i\rangle\langle\xi_i|$) prior to any measurements.

Note also that this method does not check for validity of supplied POVMs or the completeness of sets of observables, nor does it renormalize the post-measurement state.

Parameters

- ▶ `*args` – Variable-length argument list, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).

- ▶ **operators** (`list[mat | arr | QuantumObject]`) – The operator(s) with which to perform the measurement. These would typically be a (complete) set of Kraus operators forming a POVM, a (complete) set of (orthogonal) projectors forming a PVM, or a set of observables constituting a complete basis for the relevant state space.
- ▶ **observable** (`bool`) – Whether to treat the items in `operators` as observables (as opposed to Kraus operators or projectors). Defaults to `False`.
- ▶ ****kwargs** – Arbitrary keyword arguments, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).

 **Note:**

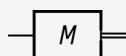
Measurement operations in quantum physics are, in general, non-linear and non-unitary operations on (normalized) state vectors and density operators. As such, they cannot be represented by matrices, and so the `matrix` property therefore does not return a valid representation of the measurement operation. Instead, it returns an identity matrix of the appropriate size for its number of dimensions and systems.

 **Note:**

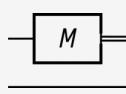
The `targets` argument must be specified as a list of numerical indices of the subsystem(s) to be measured. These indices must be consecutive, and their number must match the number of systems spanned by all given operators.

Examples

```
>>> from qhronology.mechanics.matrices import ket
>>> basis_vectors = [ket(i) for i in [0, 1]]
>>> M_basis = Measurement(operators=basis_vectors, observable=False)
>>> M_basis.diagram()
```



```
>>> pauli_matrices = [Pauli(index=i) for i in [1, 2, 3]]
>>> M_pauli = Measurement(operators=pauli_matrices, observable=True, targets=[0], num_systems=2)
>>> M_pauli.diagram()
```



```
>>> from qhronology.mechanics.matrices import ket
>>> plus = (ket(0) + ket(1)) / sp.sqrt(2)
>>> minus = (ket(0) - ket(1)) / sp.sqrt(2)
>>> M_pm = Measurement(operators=[plus, minus], observable=False, targets=[1], num_systems=2)
>>> M_pm.diagram()
```



property operators: `list[MutableDenseMatrix | ndarray | QuantumObject]`

The operator(s) with which to perform the measurement.

property observable: `bool`

Whether to treat the items in the `operators` property as observables (as opposed to Kraus operators or projectors).

9.3 Combining gates

```
class GateInterleave(
    *gates: QuantumGate (page 105),
    merge: bool | None = None,
    conjugate: bool | None = None,
    exponent: Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol
    | str | None = None,
    coefficient: Number | generic | Basic | MatrixSymbol | MatrixElement | Sym-
    bol | str | None = None,
    label: str | None = None,
    notation: str | None = None,
)
```

Bases: `QuantumGate` (page 105)

Compose two or more `QuantumGate` (page 105) instances together by interleaving them.

This is achieved by multiplying the matrix representations of the gates together. For example, for gates described by the multipartite operators $\hat{A} \otimes \hat{I}$ and $\hat{I} \otimes \hat{B}$, their interleaved composition is

$$(\hat{A} \otimes \hat{I}) \cdot (\hat{I} \otimes \hat{B}) = \hat{A} \otimes \hat{B}. \quad (9.18)$$

While this is a subclass of `QuantumGate` (page 105), all of its inherited properties, except for those corresponding to arguments in its constructor, are read-only. This is because they are calculated from their corresponding properties in the individual instances contained within the `gates` property.

Parameters

- ▶ `*gates` (`QuantumGate` (page 105)) – Variable-length argument list of `QuantumGate` (page 105) instances to be interleaved.
- ▶ `merge` (`bool`) – Whether to merge the gates together diagrammatically. Defaults to `False`.
- ▶ `conjugate` (`bool`) – Whether to perform Hermitian conjugation on the composite gate when it is called. Defaults to `False`.
- ▶ `exponent` (`num | sym | str`) – A numerical or string representation of a scalar value to which composite gate's total matrix representation is exponentiated. Must be a non-negative integer. Defaults to `1`.
- ▶ `coefficient` (`num | sym | str`) – A numerical or string representation of a scalar value by which the composite gate's matrix representation is multiplied. Performed after exponentiation. Defaults to `1`.

- ▶ **label** (*str*) – The unformatted string used to represent the gate in mathematical expressions. Defaults to " \otimes ".join([gate.label for gate in [*gates]]).
- ▶ **notation** (*str*) – The formatted string used to represent the gate in mathematical expressions. When not `None`, overrides the value passed to `label`. Not intended to be set by the user in most cases. Defaults to `None`.

Note:

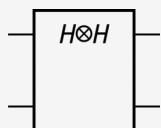
Care should be taken to ensure that gates passed to this class all have the same `num_systems` value and do not have overlapping `targets`, `controls`, and `anticontrols` properties.

Note:

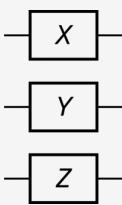
The resulting visualization (using the `diagram()` method or in circuit diagrams) may not be accurate in every case. However, the composed matrix should still be correct.

Examples

```
>>> HI = Hadamard(targets=[0], num_systems=2)
>>> IH = Hadamard(targets=[1], num_systems=2)
>>> HH = GateInterleave(HI, IH, merge=True)
>>> HH.output()
Matrix([
[1/2, 1/2, 1/2, 1/2],
[1/2, -1/2, 1/2, -1/2],
[1/2, 1/2, -1/2, -1/2],
[1/2, -1/2, -1/2, 1/2]])
```



```
>>> XII = Pauli(index=1, targets=[0], num_systems=3)
>>> IYI = Pauli(index=2, targets=[1], num_systems=3)
>>> IIZ = Pauli(index=3, targets=[2], num_systems=3)
>>> XYZ = GateInterleave(XII, IYI, IIZ)
>>> XYZ.output()
Matrix([
[0, 0, 0, 0, 0, 0, -I, 0],
[0, 0, 0, 0, 0, 0, 0, I],
[0, 0, 0, 0, I, 0, 0, 0],
[0, 0, 0, 0, 0, -I, 0, 0],
[0, 0, -I, 0, 0, 0, 0, 0],
[0, 0, 0, I, 0, 0, 0, 0],
[I, 0, 0, 0, 0, 0, 0, 0],
[0, -I, 0, 0, 0, 0, 0, 0]])
```



```
>>> CNII = Not(targets=[1], controls=[0], num_systems=4)
>>> IINC = Not(targets=[2], controls=[3], num_systems=4)
>>> CNNC = GateInterleave(CNII, IINC)
>>> CNNC.diagram()
```



property gates: list[*QuantumGate* (page 105)]

Variable-length list of *QuantumGate* (page 105) instances to be composited.

property merge: bool

Whether to merge the gates together diagrammatically.

```
class GateStack(
    *gates: QuantumGate (page 105),
    merge: bool | None = None,
    conjugate: bool | None = None,
    exponent: Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol
    | str | None = None,
    coefficient: Number | generic | Basic | MatrixSymbol | MatrixElement | Sym-
    bol | str | None = None,
    label: str | None = None,
    notation: str | None = None,
)
```

Bases: *GateInterleave* (page 133)

Compose two or more *QuantumGate* (page 105) instances together by “stacking” them vertically.

This is achieved by computing the tensor product matrix representations of the gates together. For example, for gates described by the multipartite operators $\hat{A} \otimes \hat{I}$ and $\hat{I} \otimes \hat{B}$, their stacked composition is

$$(\hat{A} \otimes \hat{I}) \otimes (\hat{I} \otimes \hat{B}) = \hat{A} \otimes \hat{I} \otimes \hat{I} \otimes \hat{B}. \quad (9.19)$$

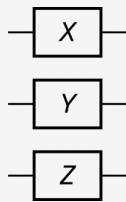
This class is derived from the *QuantumGate* (page 105) class, and so should be used in much the same way.

Parameters

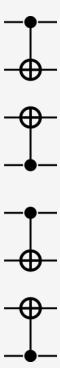
- ▶ ***gates** ([QuantumGate](#) (page 105)) – Variable-length argument list of [QuantumGate](#) (page 105) instances to be stacked.
- ▶ **merge** ([bool](#)) – Whether to merge the gates together diagrammatically. Defaults to [False](#).
- ▶ **conjugate** ([bool](#)) – Whether to perform Hermitian conjugation on the composite gate when it is called. Defaults to [False](#).
- ▶ **exponent** ([num](#) | [sym](#) | [str](#)) – A numerical or string representation of a scalar value to which composite gate's total matrix representation is exponentiated. Defaults to [1](#).
- ▶ **coefficient** ([num](#) | [sym](#) | [str](#)) – A numerical or string representation of a scalar value by which the composite gate's matrix representation is multiplied. Performed after exponentiation. Defaults to [1](#).
- ▶ **label** ([str](#)) – The unformatted string used to represent the gate in mathematical expressions. Defaults to `"⊗".join([gate.label for gate in [*gates]])`.
- ▶ **notation** ([str](#)) – The formatted string used to represent the gate in mathematical expressions. When not [None](#), overrides the value passed to **label**. Not intended to be set by the user in most cases. Defaults to [None](#).

Examples

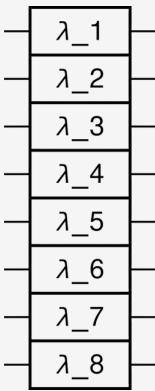
```
>>> X = Pauli(index=1)
>>> Y = Pauli(index=2)
>>> Z = Pauli(index=3)
>>> XYZ = GateStack(X, Y, Z)
>>> XYZ.output()
Matrix([
[0, 0, 0, 0, 0, 0, -I, 0],
[0, 0, 0, 0, 0, 0, 0, I],
[0, 0, 0, 0, I, 0, 0, 0],
[0, 0, 0, 0, 0, -I, 0, 0],
[0, 0, -I, 0, 0, 0, 0, 0],
[0, 0, 0, I, 0, 0, 0, 0],
[I, 0, 0, 0, 0, 0, 0, 0],
[0, -I, 0, 0, 0, 0, 0, 0]])
>>> XYZ.diagram(sep=(1, 2))
```



```
>>> gates = [Not(targets=[(i + 1) % 2], controls=[i % 2]) for i in range(0, 4)]
>>> CNOTs = GateStack(*gates)
>>> CNOTs.diagram()
```



```
>>> gates = [GellMann(index=i) for i in range(1, 9)]
>>> L = GateStack(*gates)
>>> L.diagram(sep=(1, 1))
```



In Qhronology, quantum circuits are created as instances of the [QuantumCircuit](#) (page 139) class:

```
from qhronology.quantum.circuits import QuantumCircuit
```

In the circuit diagram picturalism of these structures, time increases from left to right, and so the preparation of *input* states (created as instances of the [QuantumState](#) (page 83) class and its derivatives) begins in the past (on the left) while the measurement (or postselection) of *output* states occurs in the future (on the right). Operations on these states are represented by quantum gates (created as instances of the various subclasses of the [QuantumGate](#) (page 105) base class), and all of these events are connected by quantum wires describing the flow of quantum information (i.e., quantum probabilities) through time.

10.1 Main class

```
class QuantumCircuit(
    inputs: list[QuantumState (page 83)] | None = None,
    gates: list[QuantumGate (page 105)] | None = None,
    traces: list[int] | None = None,
    postselections: list[tuple[MutableDenseMatrix | ndarray | QuantumObject,
        int | list[int]]] | None = None,
    symbols: dict[MatrixSymbol | MatrixElement | Symbol | str, dict[str, Any]]
        | None = None,
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement
        | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement
        | Symbol | str]] | None = None,
)
)
```

Bases: [SymbolicsProperties](#)

A class for creating quantum circuits and storing their metadata.

Instances provide complete descriptions of quantum circuits, along with various associated attributes (such as mathematical conditions, including normalization). The circuit's input is recorded as a list of [QuantumState](#) (page 83) objects, with the composition of the elements of this forming the total input. Similarly, the circuit's transformation on its input is recorded as a list of [QuantumGate](#) (page 105) objects, with the product of the (linear) elements of this list forming the total transformation (e.g., unitary matrix).

Parameters

- ▶ **inputs** ([list\[QuantumState \(page 83\)\]](#)) – An ordered list of [QuantumState](#) (page 83) instances. The total input state is the tensor product of these individual states in the order in which they appear in **inputs**. Must all have the same value of the **dim** property. Defaults to `[]`.
- ▶ **gates** ([list\[QuantumGate \(page 105\)\]](#)) – An ordered list of [QuantumGate](#) (page 105) instances. The total gate is the product of these individual gates in the order in which they appear in **gates**. Must all have the same values of the **dim** and **num_systems** properties. Defaults to `[]`.
- ▶ **traces** ([list\[int\]](#)) – The numerical indices of the subsystems to be traced over. Defaults to `[]`.
- ▶ **postselections** ([list\[tuple\[mat | arr | QuantumObject, int | list\[int\]\]\]](#)) – A list of 2-tuples of vectors or matrix operators paired with the first (smallest) index of their postselection target systems. Must all have the same value of the **dim** property. Defaults to `[]`.

- ▶ **symbols** (`dict[sym | str, dict[str, Any]]`) – A dictionary in which the keys are individual symbols and the values are dictionaries of their respective SymPy keyword-argument `assumptions`. The value of the `symbols` property of all states in `inputs` and gates in `gates` are automatically merged into the instance's corresponding `symbols` property. Defaults to `{}`.
- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – A list of 2-tuples of conditions to be applied to all objects (such as states and gates) computed from the circuit. All instances of the expression in each tuple's first element are replaced by the expression in the respective second element. This uses the same format as the SymPy `subs()` method. The order in which they are applied is simply their order in the list. The value of the `conditions` property of all states in `inputs` and gates in `gates` are automatically merged into the instance's corresponding `conditions` property. Defaults to `[]`.

Note:

All states, gates, postselections, and measurement operators recorded in the instance must share the same dimensionality (i.e., the value of the `dim` property).

Note:

The sum of the `num_systems` properties of the quantum states in `inputs` should match that of each of the gates in `gates`.

Examples

Listing 10.1: Bit-flip

```
from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Not
from qhronology.quantum.circuits import QuantumCircuit

# Input
input_state = VectorState(spec=[("a", [0]), ("b", [1])], label="ψ")

# Gate
NOT = Not()

# Circuit
bitflip = QuantumCircuit(inputs=[input_state], gates=[NOT])
bitflip.diagram(pad=(0, 0), sep=(1, 1), style="unicode")

# Output
output_state = bitflip.state(label="ψ'")

# Results
input_state.print()
output_state.print()
```

```
>>> bitflip.diagram(pad=(0, 0), sep=(1, 1), style="unicode")
```

```
>>> input_state.print()
|ψ⟩ = a|0⟩ + b|1⟩
```

```
>>> output_state.print()
|ψ'⟩ = b|0⟩ + a|1⟩
```

Listing 10.2: Arbitrary state generation

```
from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Rotation, Diagonal
from qhronology.quantum.circuits import QuantumCircuit

# Input
zero_state = VectorState(spec=[(1, [0])], label="0")

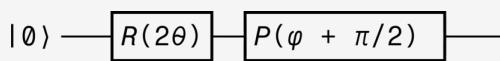
# Gates
R = Rotation(axis=1, angle="2*θ", symbols={"θ": {"real": True}}, label="R(2θ)")
P = Diagonal(
    entries={1: "φ + pi/2"}, exponentiation=True,
    symbols={"φ": {"real": True}}, label="P(φ + π/2)",
)

# Circuit
generator = QuantumCircuit(inputs=[zero_state], gates=[R, P])
generator.diagram(pad=(0, 0), sep=(1, 1), style="unicode")

# Output
arbitrary_state = generator.state(label="ψ")
arbitrary_state.simplify()

# Results
arbitrary_state.print()
```

```
>>> generator.diagram(pad=(0, 0), sep=(1, 1), style="unicode")
```



```
>>> arbitrary_state.print()
|ψ⟩ = cos(θ)|0⟩ + exp(I*φ)*sin(θ)|1⟩
```

Listing 10.3: CNOTs equivalent to SWAP

```
from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Not
from qhronology.quantum.circuits import QuantumCircuit

# Input
psi = VectorState(
    spec=[("a", [0]), ("b", [1])],
    conditions=[("a*conjugate(a) + b*conjugate(b)", 1)],
```

(continues on next page)

(continued from previous page)

```

        label=" $\psi$ ",
    )
phi = VectorState(
    spec=[("c", [0]), ("d", [1])],
    conditions=[("c*conjugate(c) + d*conjugate(d)", 1)],
    label=" $\phi$ ",
)

# Gates
CN = Not(targets=[1], controls=[0])
NC = Not(targets=[0], controls=[1])

# Circuit
swapcnots = QuantumCircuit(inputs=[psi, phi], gates=[CN, NC, CN])
swapcnots.diagram(pad=(0, 0), sep=(1, 1), style="unicode")

# Output
print(repr(swapcnots.gate()))
upper = swapcnots.state(traces=[1], label=" $\psi$ ")
lower = swapcnots.state(traces=[0], label=" $\phi$ ")
upper.kind = "pure"
lower.kind = "pure"
upper.simplify()
lower.simplify()

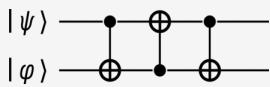
# Results
psi.print()
upper.print()
phi.print()
lower.print()

print(upper.distance(phi))
print(lower.distance(psi))

print(upper.fidelity(phi))
print(lower.fidelity(psi))

```

```
>>> swapcnots.diagram(pad=(0, 0), sep=(1, 1), style="unicode")
```



```
>>> psi.print()
|ψ⟩ = a|0⟩ + b|1⟩
```

```
>>> upper.print()
|ψ'⟩⟨ψ'| = c*conjugate(c)|0⟩⟨0| + c*conjugate(d)|0⟩⟨1| + d*conjugate(c)|1⟩⟨0| + _  
-d*conjugate(d)|1⟩⟨1|
```

```
>>> phi.print()
|φ'⟩ = c|0⟩ + d|1⟩
```

```
>>> lower.print()
|φ'⟩⟨φ'| = a*conjugate(a)|0⟩⟨0| + a*conjugate(b)|0⟩⟨1| + b*conjugate(a)|1⟩⟨0| +
    ↵ b*conjugate(b)|1⟩⟨1|
```

```
>>> upper.distance(phi)
0
```

```
>>> lower.distance(psi)
0
```

```
>>> upper.fidelity(phi)
1
```

```
>>> lower.fidelity(psi)
1
```

Listing 10.4: Bell postselection

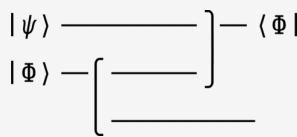
```
from qhronology.quantum.states import VectorState
from qhronology.quantum.circuits import QuantumCircuit

# Input
input_state = VectorState(
    spec=[("a", [0]), ("b", [1])],
    conditions=[("a*conjugate(a) + b*conjugate(b)", 1)],
    label="ψ",
)
bell = VectorState(spec=[(1, [0, 0]), (1, [1, 1])], norm=False, label="Φ")
```

```
# Circuit
postselection = QuantumCircuit(
    inputs=[input_state, bell], gates=[], postselections=[(bell, [0, 1])])
postselection.diagram(pad=(0, 0), sep=(4, 1), style="unicode")
```

```
# Output
output_state = postselection.state(label="ψ'")
input_state.print()
output_state.print()
```

```
>>> postselection.diagram(pad=(0, 0), sep=(4, 1), style="unicode")
```



```
>>> input_state.print()
|ψ⟩ = a|0⟩ + b|1⟩
```

```
>>> output_state.print()
|ψ'⟩ = a|0⟩ + b|1⟩
```

Listing 10.5: Fourier transform decomposition

```

from qhrontology.quantum.states import MatrixState
from qhrontology.quantum.gates import Hadamard, Phase, Fourier
from qhrontology.quantum.circuits import QuantumCircuit

import sympy as sp

size = 4 # Adjust the number of qudits
dim = 2 # Adjust the dimensionality of the Fourier transform

# Input
rho = sp.MatrixSymbol("ρ", dim**size, dim**size).asMutable()
input_state = MatrixState(spec=rho, dim=dim, label="ρ")

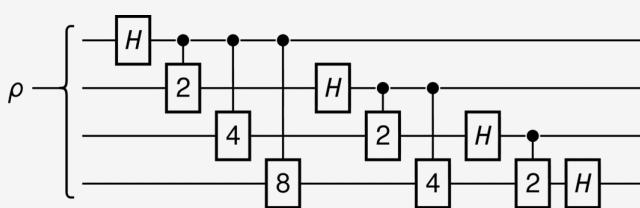
# Gates
QFT = []
for i in range(0, size):
    count = size - i
    for j in range(0, count):
        if j == 0:
            QFT.append(Hadamard(targets=[i], dim=dim, num_systems=size))
        else:
            QFT.append(
                Phase(
                    targets=[i + j],
                    controls=[i],
                    exponent=sp.Rational(1, dim**j),
                    dim=dim,
                    num_systems=size,
                    label=f"dim**{j}",
                    family="GATE",
                )
            )

# Circuit
fourier = QuantumCircuit(inputs=[input_state], gates=QFT)
fourier.diagram(pad=(0, 0), sep=(0, 1), style="unicode")

# Output
print(repr(fourier.gate()))

```

```
>>> fourier.diagram(pad=(0, 0), sep=(0, 1), style="unicode")
```



10.1.1 Constructor argument properties

property `QuantumCircuit.inputs: list[QuantumState (page 83)]`

An ordered list of `QuantumState` (page 83) instances.

The total input state is the tensor product of these individual states in the order in which they appear in the list.

Each state's `symbols` and `conditions` properties are merged into their counterparts in the instance upon their addition to the `gates` property.

property `QuantumCircuit.gates: list[QuantumGate (page 105)]`

An ordered list of `QuantumGate` (page 105) instances.

The total gate is the product of these individual gates in the order in which they appear in the list.

Must all have the same `num_systems` property.

Each gate's `symbols` and `conditions` properties are merged into their counterparts in the instance upon their addition to the `gates` property.

property `QuantumCircuit.traces: list[int]`

The numerical indices of the subsystems to be traced over.

property `QuantumCircuit.postselections: list[tuple[MutableDenseMatrix | ndarray | QuantumObject, int | list[int]]]`

A list of 2-tuples of vectors or matrix operators paired with the first (smallest) index of their postselection target systems.

Any `symbols` and `conditions` properties of each postselection are merged into their counterparts in the instance upon their addition to the `postselections` property.

property `QuantumCircuit.symbols: dict[MatrixSymbol | MatrixElement | Symbol | str, dict[str, Any]]`

A dictionary in which the keys are individual symbols (contained within the object's matrix representation) and the values are dictionaries of their respective SymPy keyword-argument `assumptions` ("predicates"). A full list of currently supported predicates, and their defaults, is as follows:

- ▶ "algebraic": True
- ▶ "commutative": True
- ▶ "complex": True
- ▶ "extended_negative": False
- ▶ "extended_nonnegative": True
- ▶ "extended_nonpositive": False
- ▶ "extended_nonzero": True
- ▶ "extended_positive": True
- ▶ "extended_real": True
- ▶ "finite": True
- ▶ "hermitian": True
- ▶ "imaginary": False

- ▶ "infinite": False
 - ▶ "integer": True
 - ▶ "irrational": False
 - ▶ "negative": False
 - ▶ "noninteger": False
 - ▶ "nonnegative": True
 - ▶ "nonpositive": False
 - ▶ "nonzero": True
 - ▶ "positive": True
 - ▶ "rational": True
 - ▶ "real": True
 - ▶ "transcendental": False
 - ▶ "zero": False
-

property `QuantumCircuit.conditions:` `list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]]`

A list of 2-tuples of conditions to be applied to the object's matrix representation.

10.1.2 Read-only properties

property `QuantumCircuit.dim:` `int`

The dimensionality of the circuit. Calculated from its states and gates, and so all must have the same value.

property `QuantumCircuit.systems:` `list[int]`

An ordered list of the numerical indices of the circuit's systems.

property `QuantumCircuit.systems_traces:` `list[int]`

The indices of the systems to be traced over.

property `QuantumCircuit.systems_postselections:` `list[int]`

The indices of the systems to be postselected.

property `QuantumCircuit.systems_removed:` `list[int]`

The indices of all of the systems targeted by the `traces` and `postselections` properties.

property `QuantumCircuit.num_systems:` `int`

Alias for `num_systems_gross`.

`property QuantumCircuit.num_systems_inputs: int`

The total number of systems spanned by the circuit's input states.

`property QuantumCircuit.num_systems_gates: int`

The total number of systems spanned by the circuit's gates.

`property QuantumCircuit.num_systems_gross: int`

The total number of systems spanned by the circuit's states and gates prior to any system reduction (post-processing, i.e., traces and postselections]).

`property QuantumCircuit.num_systems_net: int`

The total number of systems spanned by the circuit's states and gates after any system reduction (post-processing, i.e., traces and postselections]).

`property QuantumCircuit.num_systems_removed: int`

The total number of systems removed via system reduction (post-processing, i.e., traces and postselections]).

`property QuantumCircuit.input_is_vector: bool`

Whether all states in `inputs` are vector states.

`property QuantumCircuit.gate_is_linear: bool`

Whether all gates are linear (i.e., not measurement operations).

`property QuantumCircuit.post_is_vector: bool`

Whether any traces or non-vector postselections exist in the circuit's post-processing (trace and postselection) stage.

`property QuantumCircuit.output_is_vector: bool`

Whether or not the output from the entire circuit is a vector state.

`property QuantumCircuit.matrix: MutableDenseMatrix`

The matrix representation of the total output state prior to any post-processing (i.e., traces and postselections).

10.1.3 Methods

```
QuantumCircuit.input(
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
    simplify: bool | None = None,
    conjugate: bool | None = None,
    norm: bool | Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str | None = None,
```

```

label: str | None = None,
notation: str | None = None,
debug: bool | None = None,
) → QuantumState (page 83)

```

Construct the composite input state of the quantum circuit as a [QuantumState](#) (page 83) instance and return it.

This is computed as the tensor product of the individual gates in the order in which they appear in the `inputs` property. Is a vector state only when all of the component states are vectors.

Parameters

- ▶ `conditions` (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ `simplify` (`bool`) – Whether to perform algebraic simplification on the state. Defaults to `False`.
- ▶ `conjugate` (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ `norm` (`bool | num | sym | str`) – The value to which the state is normalized. If `True`, normalizes to a value of 1. If `False`, does not normalize. Defaults to `False`.
- ▶ `label` (`str`) – The unformatted string used to represent the state in mathematical expressions. Must have a non-zero length. Defaults to `"⊗".join([state.label for state in self.inputs])`.
- ▶ `notation` (`str`) – The formatted string used to represent the state in mathematical expressions. When not `None`, overrides the value passed to `label`. Must have a non-zero length. Not intended to be set by the user in most cases. Defaults to `None`.
- ▶ `debug` (`bool`) – Whether to print the internal state (held in `matrix`) on change. Defaults to `False`.

Returns

`mat` – The total input state as a [QuantumState](#) (page 83) instance.

```

QuantumCircuit.gate(
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
    simplify: bool | None = None,
    conjugate: bool | None = None,
    exponent: Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str | None = None,
    label: str | None = None,
    notation: str | None = None,
) → QuantumGate (page 105)

```

Construct the combined gate describing the total sequence of gates in the quantum circuit as a [QuantumGate](#) (page 105) instance and return it.

This is computed as the matrix product of the individual gates in the reverse order in which they appear in the `gates` property.

Parameters

- ▶ `conditions` (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the gate. Defaults to the value of `self.conditions`.
- ▶ `simplify` (`bool`) – Whether to perform algebraic simplification on the gate. Defaults to `False`.
- ▶ `conjugate` (`bool`) – Whether to perform Hermitian conjugation on the gate when it is called. Defaults to `False`.

- ▶ **exponent** (`num | sym | str`) – A numerical or string representation of a scalar value to which gate's operator (residing on `targets`) is exponentiated. Must be a non-negative integer. Defaults to `1`.
- ▶ **label** (`str`) – The unformatted string used to represent the gate in mathematical expressions. Defaults to `"U"`.
- ▶ **notation** (`str`) – The formatted string used to represent the gate in mathematical expressions. When not `None`, overrides the value passed to `label`. Not intended to be set by the user in most cases. Defaults to `None`.

Returns

`mat` – The matrix or vector representation of the total gate sequence.

Note:

This construction excludes measurement gates as they do not have a corresponding matrix representation.

QuantumCircuit.output()

```
conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
simplify: bool | None = None,
conjugate: bool | None = None,
postprocess: bool | None = None,
) → MutableDenseMatrix
```

Compute the matrix representation of the total output state of the circuit (including any post-processing, i.e., traces and postselections) and return it.

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the state. Defaults to `False`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ **postprocess** (`bool`) – Whether to post-process the state (i.e., perform the circuit's traces and postselections). Defaults to `True`.

Returns

`mat` – The matrix representation of the (post-processed) output state.

QuantumCircuit.state()

```
conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
simplify: bool | None = None,
conjugate: bool | None = None,
norm: bool | Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str | None = None,
label: str | None = None,
notation: str | None = None,
traces: list[int] | None = None,
postprocess: bool | None = None,
debug: bool | None = None,
) → QuantumState (page 83)
```

Compute the total output state of the circuit (including any post-processing, i.e., traces and postselections) as a [QuantumState](#) (page 83) instance and return it.

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the state before committing it to the `matrix` property. Defaults to `False`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ **norm** (`bool | num | sym | str`) – The value to which the state is normalized. If `True`, normalizes to a value of 1. If `False`, does not normalize. Defaults to `False`.
- ▶ **label** (`str`) – The unformatted string used to represent the state in mathematical expressions. Must have a non-zero length. Defaults to " `ρ` " (if `form == "matrix"`) or " `ψ` " (if `form == "vector"`).
- ▶ **notation** (`str`) – The formatted string used to represent the state in mathematical expressions. When not `None`, overrides the value passed to `label`. Must have a non-zero length. Not intended to be set by the user in most cases. Defaults to `None`.
- ▶ **traces** (`list[int]`) – A list of indices of the systems (relative to the entire circuit) on which to perform partial traces. Performed regardless of the value of `postprocess`. Defaults to `[]`.
- ▶ **postprocess** (`bool`) – Whether to post-process the state (i.e., perform the circuit's traces and postselections). Defaults to `True`.
- ▶ **debug** (`bool`) – Whether to print the internal state (held in `matrix`) on change. Defaults to `False`.

Returns

`QuantumState` – The (post-processed) output state as a [QuantumState](#) (page 83) instance.

```
QuantumCircuit.measure(
    operators: list[MutableDenseMatrix | ndarray | QuantumObject],
    targets: int | list[int] | None = None,
    observable: bool | None = None,
    statistics: bool | None = None,
) → QuantumState (page 83) | list[Number | generic | Basic | MatrixSymbol |
    MatrixElement | Symbol]
```

Perform a quantum measurement on one or more systems (indicated in `targets`) of the circuit's total output state. This occurs prior to any post-processing (i.e., traces and postselections).

This method has two main modes of operation:

- ▶ When `statistics` is `True`, the (reduced) state ($\hat{\rho}$) (residing on the systems indicated in `targets`) is measured and the set of resulting statistics is returned. This takes the form of an ordered list of values $\{p_i\}_i$ associated with each given operator, where:
 - $p_i = \text{tr}[\hat{K}_i^\dagger \hat{K}_i \hat{\rho}]$ (measurement probabilities) when `observable` is `False` (`operators` is a list of Kraus operators or projectors \hat{K}_i)
 - $p_i = \text{tr}[\hat{O}_i \hat{\rho}]$ (expectation values) when `observable` is `True` (`operators` is a list of observables \hat{O}_i)
- ▶ When `statistics` is `False`, the (reduced) state ($\hat{\rho}$) (residing on the systems indicated in `targets`) is measured and mutated it according to its predicted post-measurement form (i.e., the sum of all possible measurement outcomes). This yields the transformed states:
 - When `observable` is `False`:

$$\hat{\rho}' = \sum_i \hat{K}_i \hat{\rho} \hat{K}_i^\dagger. \quad (10.1)$$

- When `observable` is `True`:

$$\hat{\rho}' = \sum_i \text{tr}[\hat{O}_i \hat{\rho}] \hat{O}_i. \quad (10.2)$$

In the case where `operators` contains only a single item (\hat{K}), and the current state ($|\psi\rangle$) is a vector form, the transformation of the state is in accordance with the rule

$$|\psi'\rangle = \frac{\hat{K}|\psi\rangle}{\sqrt{\langle\psi|\hat{K}^\dagger\hat{K}|\psi\rangle}} \quad (10.3)$$

when `observable` is `False`. In all other mutation cases, the post-measurement state is a matrix, even if the pre-measurement state was a vector.

The items in the list `operators` can also be vectors (e.g., $|\xi_i\rangle$), in which case each is converted into its corresponding operator matrix representation (e.g., $|\xi_i\rangle\langle\xi_i|$) prior to any measurements.

Parameters

- ▶ `operators` (`list[mat | arr | QuantumObject]`) – The operator(s) with which to perform the measurement. These would typically be a (complete) set of Kraus operators forming a POVM, a (complete) set of (orthogonal) projectors forming a PVM, or a set of observables constituting a complete basis for the relevant state space.
- ▶ `targets` (`int | list[int]`) – The numerical indices of the system(s) to be measured. They must be consecutive, and their number must match the number of systems spanned by all given operators. Indexing begins at `0`. All other systems are discarded (traced over) in the course of performing the measurement.
- ▶ `observable` (`bool`) – Whether to treat the items in `operators` as observables (as opposed to Kraus operators or projectors). Defaults to `False`.
- ▶ `statistics` (`bool`) – Whether to return a list of probabilities (`True`) or mutate the state into a post-measurement probabilistic sum of all outcomes (`False`). Defaults to `False`.

Returns

- ▶ `list[num / sym]` – A list of probabilities corresponding to each operator given in `operators`. Returned only if `statistics` is `True`.
- ▶ `QuantumState` – A quantum state that takes the form of the post-measurement probabilistic sum of all outcomes of measurements corresponding to each operator given in `operators`. Returned only if `statistics` is `False`.

`QuantumCircuit.diagram()`

```
pad: tuple[int, int] | None = None,
sep: tuple[int, int] | None = None,
uniform_spacing: bool | None = None,
force_separation: bool | None = None,
style: str | None = None,
return_string: bool | None = None,
) → None | str
```

Print or return a diagram of the quantum circuit as a multiline string.

Parameters

- ▶ `pad` (`tuple[int, int]`) – A two-tuple describing the horizontal and vertical interior paddings between the content at the centre of each gate (e.g., label) and its outer edge (e.g., block border). Both integers must be non-negative. Defaults to `(0, 0)`.

- ▶ **sep** (`tuple[int, int]`) – A two-tuple describing the horizontal and vertical exterior separation distances between the edges of neighbouring gates. Both integers must be non-negative. Defaults to `(1, 1)`.
- ▶ **uniform_spacing** (`bool`) – Whether to uniformly space the gates horizontally such that the midpoint of each is equidistant from those of its neighbours. Defaults to `False`.
- ▶ **force_separation** (`bool`) – Whether to force the horizontal gate separation to be exactly the value given in `sep` for all gates in the circuit. When not `False`, the value of `uniform_spacing` is ignored. Defaults to `False`.
- ▶ **style** (`str`) – A string specifying the style for the circuit visualization to take. Can be any of `"ascii"`, `"unicode"`, or `"unicode_alt"`. Defaults to `"unicode"`.
- ▶ **return_string** (`bool`) – Whether to return the assembled diagram as a multiline string. Defaults to `False`.

Returns

- ▶ `None` – Returned only if `return_string` is `False`.
- ▶ `str` – The rendered circuit diagram. Returned only if `return_string` is `True`.

Note:

The quality of the visualization depends greatly on the output's configuration. For best results, the terminal should have a monospace font with good Unicode coverage.

Examples

```
from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Hadamard, Not, Measurement, Pauli,_
    GateInterleave
from qhronology.quantum.circuits import QuantumCircuit
from qhronology.mechanics.matrices import ket

# Input
teleporting_state = VectorState(
    spec=[["a", "b"]],
    symbols={"a": {"complex": True}, "b": {"complex": True}},
    conditions=[("a*conjugate(a) + b*conjugate(b)", "1")],
    label="ψ",
)
zero_state = VectorState(spec=[(1, [0, 0])], label="0,0")

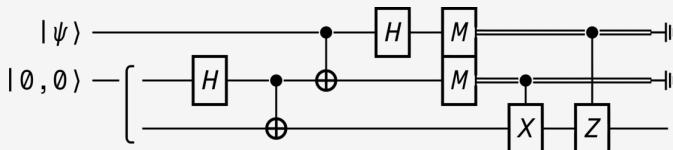
# Gates
IHI = Hadamard(targets=[1], num_systems=3)
ICN = Not(targets=[2], controls=[1], num_systems=3)
CNI = Not(targets=[1], controls=[0], num_systems=3)
HII = Hadamard(targets=[0], num_systems=3)
IMI = Measurement(
    operators=[ket(0), ket(1)], observable=False, targets=[1], num_systems=3
)
MII = Measurement(
    operators=[ket(0), ket(1)], observable=False, targets=[0], num_systems=3
)
MMI = GateInterleave(MII, IMI)
ICX = Pauli(index=1, targets=[2], controls=[1], num_systems=3)
CIZ = Pauli(index=3, targets=[2], controls=[0], num_systems=3)
```

(continues on next page)

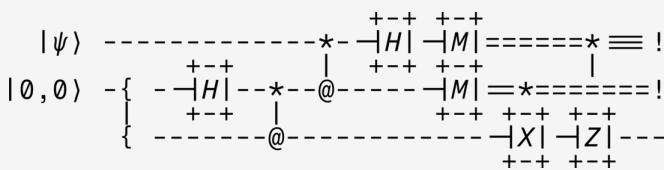
(continued from previous page)

```
# Circuit
circuit = QuantumCircuit(
    inputs=[teleporting_state, zero_state],
    gates=[IHI, ICN, CNI, HII, MMI, ICX, CIZ],
    traces=[0, 1],
)
```

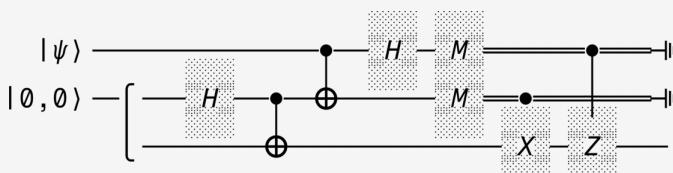
```
>>> circuit.diagram(pad=(0, 0), sep=(1, 1), style="unicode")
```



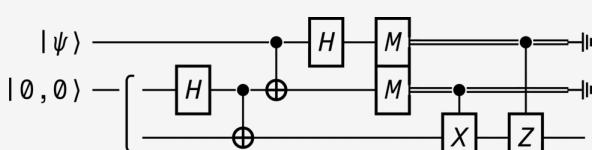
```
>>> circuit.diagram(pad=(0, 0), sep=(1, 1), style="ascii")
```



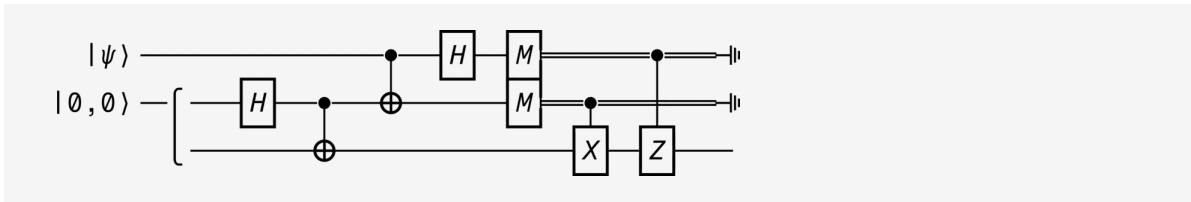
```
>>> circuit.diagram(pad=(0, 0), sep=(1, 1), style="unicode_alt")
```



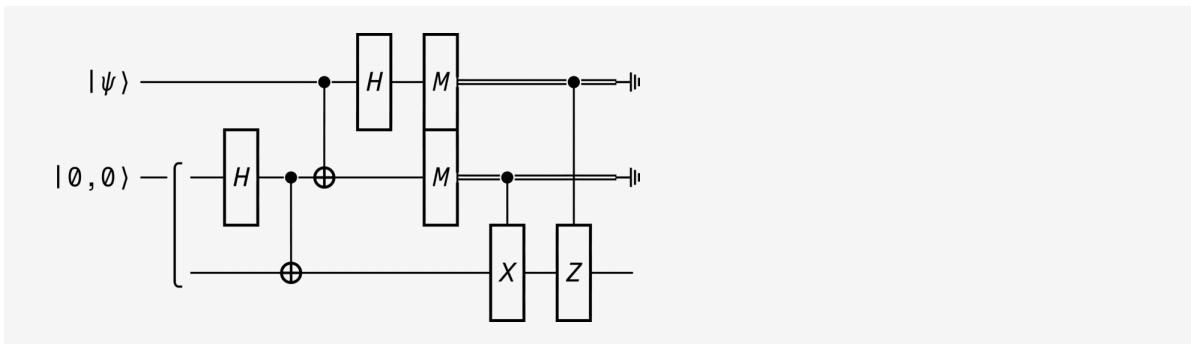
```
>>> circuit.diagram(pad=(0, 0), sep=(1, 1), force_separation=True, style="unicode")
```



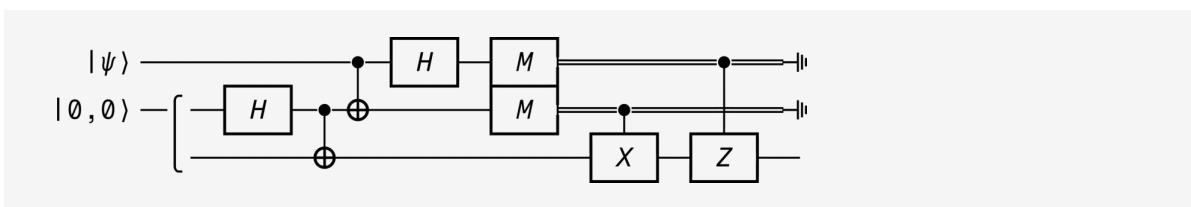
```
>>> circuit.diagram(pad=(0, 0), sep=(1, 1), uniform_spacing=True, style="unicode")
```



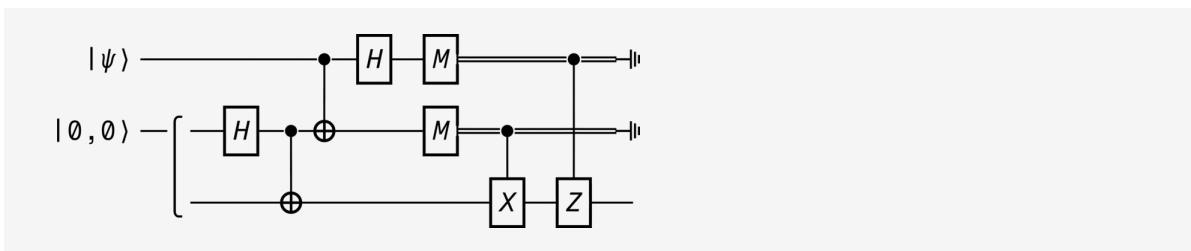
```
>>> circuit.diagram(pad=(0, 1), sep=(1, 1), force_separation=True, style="unicode")
```



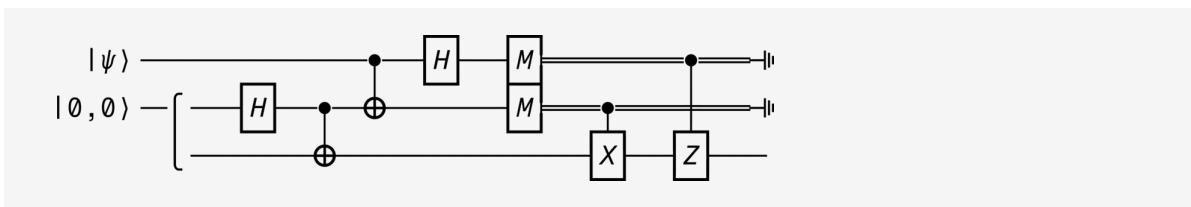
```
>>> circuit.diagram(pad=(1, 0), sep=(1, 1), force_separation=True, style="unicode")
```



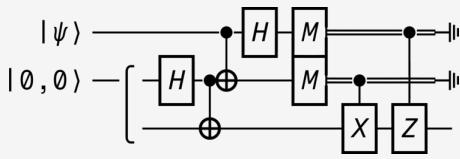
```
>>> circuit.diagram(pad=(0, 0), sep=(1, 2), force_separation=True, style="unicode")
```



```
>>> circuit.diagram(pad=(0, 0), sep=(2, 1), force_separation=True, style="unicode")
```



```
>>> circuit.diagram(pad=(0, 0), sep=(0, 1), force_separation=True, style="unicode")
```



In Qhronology, quantum circuit models of closed timelike curves (CTCs) are created as instances of the `QuantumCTC` (page 157) class:

```
from qhronology.quantum.prescriptions import QuantumCTC
```

This provides almost identical functionality as the `QuantumCircuit` (page 139) class (from which it is derived), differing only in the addition of the `systems_respecting` (page 161) and `systems_violating` (page 161) properties.

Built upon the `QuantumCTC` (page 157) class are the various theoretical models (prescriptions) of quantum time travel. Qhronology currently implements two such prescriptions: *Deutsch's model* (D-CTCs) and *postselected teleportation* (P-CTCs):

```
from qhronology.quantum.prescriptions import DCTC, PCTC
```

Due to their close relationship, instances of the `QuantumCTC` (page 157) class can be created directly from instances of the `QuantumCircuit` (page 139) class. This is achieved by way of the `circuit` argument in the `QuantumCTC` (page 157) constructor, whereby all attributes of the given `QuantumCircuit` (page 139) instance are copied (deeply) to the `QuantumCTC` (page 157) instance during initialization. Importantly, this enables the various subclasses, such as `DCTC` (page 164) and `PCTC` (page 169), to be instantiated entirely using a single `QuantumCTC` (page 157) instance, thereby reducing duplication (of arguments) when one wishes to study multiple prescriptions of the same CTC circuit.

11.1 Main class

Please note that the documentation of this class includes only properties and methods that are either new or modified from its base class `QuantumCircuit` (page 139).

```
class QuantumCTC(  
    *args,  
    circuit: QuantumCircuit (page 139) | None = None,  
    systems_respecting: list[int] | None = None,  
    systems_violating: list[int] | None = None,  
    **kwargs,  
)
```

Bases: `QuantumCircuit` (page 139)

A class for creating quantum circuit models of quantum interactions near closed timelike curves and storing their metadata.

This is built upon the `QuantumCircuit` (page 139) class, and so inherits all of its attributes, properties, and methods.

Instances provide complete descriptions of quantum circuits involving antichronological time travel. The class however does not possess any ability to compute the output state (e.g., resolve temporal paradoxes) of the circuit; this is functionality that is associated with the specific prescriptions of quantum time travel, and such prescriptions are implemented as subclasses.

Parameters

- ▶ `*args` – Variable length argument list, passed directly to the constructor `__init__` of the superclass `QuantumCircuit` (page 139).

- ▶ **circuit** ([QuantumCircuit](#) (page 139)) – An instance of the [QuantumCircuit](#) (page 139) class. The values of its attributes override any other values specified in `*args` and `**kwargs`. Defaults to `None`.
- ▶ **systems_respecting** (`int` | `list[int]`) – The numerical indices of the chronology-respecting (CR) subsystems. Defaults to `[]`.
- ▶ **systems_violating** (`int` | `list[int]`) – The numerical indices of the chronology-violating (CV) subsystems. Defaults to `[]`.
- ▶ ****kwargs** – Arbitrary keyword arguments, passed directly to the constructor `__init__` of the superclass [QuantumCircuit](#) (page 139).

i Note:

The lists of indices specified in `systems_respecting` and `systems_violating` must both be contiguous. Additionally, the circuit's inputs (`inputs`) are treated as one contiguous total state, with the indices of its subsystems exactly matching those specified in `systems_respecting`.

i Note:

It is best practice to specify only one of either `systems_violating` or `systems_respecting`, never both. The properties associated with both of these constructor arguments automatically ensure that they are always complementary (with respect to the entire system space), and so only one needs to be specified.

i Note:

The `circuit` argument can be used to merge the value of every attribute from a pre-existing [QuantumCircuit](#) (page 139) instance into the [QuantumCTC](#) (page 157) instance. Any such mergers override the values of the attributes associated with the other arguments specified in the constructor. It is best practice to specify either of:

- ▶ only `circuit` and one of either `systems_respecting` or `systems_violating`
- ▶ `*args` and `**kwargs` (like a typical initialization of a [QuantumCircuit](#) (page 139) instance) without specifying `circuit`

i Note:

The total interaction between the CR and CV systems is expected to be unitary, and so the sequence of gates in `gates` cannot contain any non-unitary gates (e.g., measurement operations).

i Note:

Post-processing (e.g., traces and postselections) cannot be performed on any chronology-violating (CV) systems (i.e., those corresponding to indices specified in `systems_violating`).

Examples

Listing 11.1: SWAP interaction with a CTC

```

from chronology.quantum.states import MixedState
from chronology.quantum.gates import Swap, Pauli
from chronology.quantum.prescriptions import QuantumCTC, DCTC, PCTC

import sympy as sp

# Input
rho = sp.MatrixSymbol("ρ", 2, 2).asMutable()
input_state = MixedState(
    spec=rho,
    conditions=[(rho[1, 1], 1 - rho[0, 0])],
    label="ρ",
)

# Gate
S = Swap(targets=[0, 1], num_systems=2)
I = Pauli(index=0, targets=[0, 1], num_systems=2)

# CTC
SWAP = QuantumCTC([input_state], [S], systems_respecting=[0])
SWAP.diagram()

# Output
# D-CTCs
SWAP_DCTC = DCTC(circuit=SWAP)
SWAP_DCTC_respecting = SWAP_DCTC.state_respecting(norm=False, label="ρ_D")
SWAP_DCTC_violating = SWAP_DCTC.state_violating(norm=False, label="τ_D")
SWAP_DCTC_respecting.conditions = [(1 - rho[0, 0], rho[1, 1])]
SWAP_DCTC_respecting.simplify()
SWAP_DCTC_violating.conditions = [(1 - rho[0, 0], rho[1, 1])]

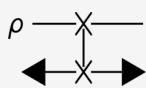
# P-CTCs
SWAP_PCTC = PCTC(circuit=SWAP)
SWAP_PCTC_respecting = SWAP_PCTC.state_respecting(norm=False, label="ρ_P")
SWAP_PCTC_violating = SWAP_PCTC.state_violating(norm=False, label="τ_P")
SWAP_PCTC_respecting.conditions = [(1 - rho[0, 0], rho[1, 1])]
SWAP_PCTC_violating.conditions = [(1 - rho[0, 0], rho[1, 1])]

SWAP_PCTC_respecting.simplify()
SWAP_PCTC_violating.simplify()

# Results
SWAP_DCTC_respecting.print()
SWAP_DCTC_violating.print()
SWAP_PCTC_respecting.print()
SWAP_PCTC_violating.print()

```

```
>>> SWAP.diagram()
```



```
>>> SWAP_DCTC_respecting.print()
rho_D = rho[0, 0]|0><0| + rho[0, 1]|0><1| + rho[1, 0]|1><0| + rho[1, 1]|1><1|
```

```
>>> SWAP_DCTC_violating.print()
tau_D = rho[0, 0]|0><0| + rho[0, 1]|0><1| + rho[1, 0]|1><0| + rho[1, 1]|1><1|
```

```
>>> SWAP_PCTC_respecting.print()
rho_P = rho[0, 0]|0><0| + rho[0, 1]|0><1| + rho[1, 0]|1><0| + rho[1, 1]|1><1|
```

```
>>> SWAP_PCTC_violating.print()
tau_P = rho[0, 0]|0><0| + rho[0, 1]|0><1| + rho[1, 0]|1><0| + rho[1, 1]|1><1|
```

Listing 11.2: CNOT interaction with a CTC

```
from qhronology.quantum.states import MixedState
from qhronology.quantum.gates import Not
from qhronology.quantum.circuits import QuantumCircuit
from qhronology.quantum.prescriptions import QuantumCTC, DCTC, PCTC

import sympy as sp

# Input
rho = sp.MatrixSymbol("rho", 2, 2).asMutable()
input_state = MixedState(spec=rho, conditions=[(rho[1, 1], 1 - rho[0, 0])], label="rho")

# Gate
CN = Not(targets=[0], controls=[1], num_systems=2)

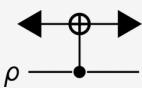
# CTC
CNOT = QuantumCircuit(inputs=[input_state], gates=[CN])
CNOT = QuantumCTC(circuit=CNOT, systems_respecting=[1])
CNOT.diagram()

# Output
# D-CTCs
CNOT_DCTC = DCTC(circuit=CNOT)
CNOT_DCTC_respecting = CNOT_DCTC.state_respecting(norm=False, label="rho_D")
CNOT_DCTC_violating = CNOT_DCTC.state_violating(norm=False, label="tau_D")
CNOT_DCTC_respecting.conditions = [(1 - rho[0, 0], rho[1, 1])]

# P-CTCs
CNOT_PCTC = PCTC(circuit=CNOT)
CNOT_PCTC_respecting = CNOT_PCTC.state_respecting(norm=True, label="rho_P")
CNOT_PCTC_violating = CNOT_PCTC.state_violating(norm=False, label="tau_P")

# Results
CNOT_DCTC_respecting.print()
CNOT_DCTC_violating.print()
CNOT_PCTC_respecting.print()
CNOT_PCTC_violating.print()

>>> CNOT.diagram(pad=(0,0), sep=(1,1), style="unicode")
```



```
>>> CNOT_DCTC_respecting.print()
rho_D = rho[0, 0]|0><0| + 2*g*rho[0, 1]|0><1| + 2*g*rho[1, 0]|1><0| + rho[1, 1]|1><1|
```

```
>>> CNOT_DCTC_violating.print()
tau_D = 1/2|0><0| + g|0><1| + g|1><0| + 1/2|1><1|
```

```
>>> CNOT_PCTC_respecting.print()
rho_P = |0><0|
```

```
>>> CNOT_PCTC_violating.print()
tau_P = 1/2|0><0| + 1/2|1><1|
```

11.1.1 Constructor argument properties

property `QuantumCTC.systems_respecting: list[int]`

The numerical indices of the chronology-respecting (CR) subsystems.

property `QuantumCTC.systems_violating: list[int]`

The numerical indices of the chronology-violating (CV) subsystems.

11.1.2 Methods

```
QuantumCTC.input(
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
    simplify: bool | None = None,
    conjugate: bool | None = None,
    norm: bool | Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str | None = None,
    label: str | None = None,
    notation: str | None = None,
    debug: bool | None = None,
) → QuantumState (page 83)
```

Construct the composite chronology-respecting (CR) input state of the closed timelike curve as a `QuantumState` (page 83) instance and return it.

This is computed as the tensor product of the individual gates in the order in which they appear in the `inputs` property. Is a vector state only when all of the component states are vectors.

Parameters

- ▶ `conditions` (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. If `False`, does not substitute the conditions. Defaults to the value of `self.conditions`.
- ▶ `simplify` (`bool`) – Whether to perform algebraic simplification on the state. Defaults to `False`.

- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ **norm** (`bool | num | sym | str`) – The value to which the state is normalized. If `True`, normalizes to a value of 1. If `False`, does not normalize. Defaults to `False`.
- ▶ **label** (`str`) – The unformatted string used to represent the state in mathematical expressions. Must have a non-zero length. Defaults to `"⊗".join([state.label for state in self.inputs])`.
- ▶ **notation** (`str`) – The formatted string used to represent the state in mathematical expressions. When not `None`, overrides the value passed to `label`. Must have a non-zero length. Not intended to be set by the user in most cases. Defaults to `None`.
- ▶ **debug** (`bool`) – Whether to print the internal state (held in `matrix`) on change. Defaults to `False`.

Returns

`mat` – The total input state as a `QuantumState` (page 83) instance.

```
QuantumCTC.diagram(  
    pad: tuple[int, int] | None = None,  
    sep: tuple[int, int] | None = None,  
    uniform_spacing: bool | None = None,  
    force_separation: bool | None = None,  
    style: str | None = None,  
    return_string: bool | None = None,  
) → None | str
```

Print or return a diagram of the quantum circuit as a multiline string.

Parameters

- ▶ **pad** (`tuple[int, int]`) – A two-tuple describing the horizontal and vertical interior paddings between the content at the centre of each gate (e.g., label) and its outer edge (e.g., block border). Both integers must be non-negative. Defaults to `(0, 0)`.
- ▶ **sep** (`tuple[int, int]`) – A two-tuple describing the horizontal and vertical exterior separation distances between the edges of neighbouring gates. Both integers must be non-negative. Defaults to `(1, 1)`.
- ▶ **uniform_spacing** (`bool`) – Whether to uniformly space the gates horizontally such that the midpoint of each is equidistant from those of its neighbours. Defaults to `False`.
- ▶ **force_separation** (`bool`) – Whether to force the horizontal gate separation to be exactly the value given in `sep` for all gates in the circuit. When not `False`, the value of `uniform_spacing` is ignored. Defaults to `False`.
- ▶ **style** (`str`) – A string specifying the style for the circuit visualization to take. Can be any of `"ascii"`, `"unicode"`, or `"unicode_alt"`. Defaults to `"unicode"`.
- ▶ **return_string** (`bool`) – Whether to return the assembled diagram as a multiline string. Defaults to `False`.

Returns

- ▶ `None` – Returned only if `return_string` is `False`.
- ▶ `str` – The rendered circuit diagram. Returned only if `return_string` is `True`.

Note:

The quality of the visualization depends greatly on the output's configuration. For best results, the terminal should have a monospace font with good Unicode coverage.

Examples

Please see the examples of the [QuantumCTC](#) (page 157) class itself. For advanced usage examples, see the corresponding [diagram\(\)](#) (page 151) method of the [QuantumCircuit](#) (page 139) class.

11.2 Subclasses

Please note that the documentation of these subclasses includes only properties and methods that are either new or modified from the base class [QuantumCTC](#) (page 157).

11.2.1 Deutsch's prescription (D-CTCs)

11.2.1.1 Functions

```
dctc_violating(
    input_respecting: MutableDenseMatrix | QuantumState (page 83),
    gate: MutableDenseMatrix | QuantumGate (page 105),
    systems_respecting: list[int],
    systems_violating: list[int],
    free_symbol: MatrixSymbol | MatrixElement | Symbol | str | None = None,
) → MutableDenseMatrix
```

Calculate the chronology-violating (CV) state(s) according to the D-CTC prescription by computing fixed points of the map

$$\mathbf{d}_{\hat{U}}[\hat{\rho}, \hat{\tau}] = \text{tr}_{\text{CR}}[\hat{U}(\hat{\rho} \otimes \hat{\tau})\hat{U}^\dagger] \quad (11.1)$$

given the chronology-respecting (CR) input state [input_respecting](#) ($\hat{\rho}$) and (unitary) interaction described by [gate](#) (\hat{U}).

Parameters

- ▶ [input_respecting](#) ([mat](#) | [QuantumState](#) (page 83)) – The matrix representation of the chronology-respecting (CR) input state.
- ▶ [gate](#) ([mat](#) | [QuantumGate](#) (page 105)) – The matrix representation of the gate describing the (unitary) interaction between the CR and CV systems.
- ▶ [systems_respecting](#) ([list\[int\]](#)) – The numerical indices of the chronology-respecting (CR) subsystems.
- ▶ [systems_violating](#) ([list\[int\]](#)) – The numerical indices of the chronology-violating (CV) subsystems.
- ▶ [free_symbol](#) ([sym](#) | [str](#)) – The representation of the algebraic symbol to be used as the free parameter in the case where the CV map has a multiplicity of fixed points. Defaults to "g".

Returns

[mat](#) – The fixed-point solution(s) of the D-CTC CV map.

Note:

Please note that this function in its current form is considered to be *highly experimental*.

```
dctc_respecting(
    input_respecting: MutableDenseMatrix | QuantumState (page 83),
    input_violating: MutableDenseMatrix | QuantumState (page 83),
    gate: MutableDenseMatrix | QuantumGate (page 105),
```

```
    systems_respecting: list[int],
    systems_violating: list[int],
) → MutableDenseMatrix
```

Calculate the chronology-respecting (CR) state(s) according to the D-CTC prescription's CR map

$$\mathbf{D}_{\hat{U}}[\hat{\rho}, \hat{\tau}] = \text{tr}_{\text{CV}}[\hat{U}(\hat{\rho} \otimes \hat{\tau})\hat{U}^\dagger] \quad (11.2)$$

given the chronology-respecting (CR) input state `input_respecting` ($\hat{\rho}$), chronology-violating (CV) solution state `input_violating` ($\hat{\tau}$), and (unitary) interaction described by `gate` (\hat{U}).

Parameters

- ▶ `input_respecting` (`mat` | `QuantumState` (page 83)) – The matrix representation of the chronology-respecting (CR) input state.
- ▶ `input_violating` (`mat` | `QuantumState` (page 83)) – The matrix representation of the chronology-violating (CR) solution state.
- ▶ `gate` (`mat` | `QuantumGate` (page 105)) – The matrix representation of the gate describing the (unitary) interaction between the CR and CV systems.
- ▶ `systems_respecting` (`list[int]`) – The numerical indices of the chronology-respecting (CR) subsystems.
- ▶ `systems_violating` (`list[int]`) – The numerical indices of the chronology-violating (CV) subsystems.

Returns

`mat` – The solution(s) of the D-CTC CR map.

11.2.1.2 Class

```
class DCTC(
    *args,
    free_symbol: MatrixSymbol | MatrixElement | Symbol | str | None = None,
    **kwargs,
)
```

Bases: `QuantumCTC` (page 157)

A subclass for creating closed timelike curves described by Deutsch's prescription (D-CTCs) of quantum time travel.

This is built upon the `QuantumCTC` (page 157) class, and so inherits all of its attributes, properties, and methods.

Parameters

- ▶ `*args` – Variable-length argument list, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).
- ▶ `free_symbol` (`sym` | `str`) – The representation of the algebraic symbol to be used as the free parameter in the case where the CV map has a multiplicity of fixed points. Defaults to "g".
- ▶ `**kwargs` – Arbitrary keyword arguments, passed directly to the constructor `__init__` of the superclass `QuantumGate` (page 105).

Examples

For usage examples, please see the superclass `QuantumCTC` (page 157).

11.2.1.2.1 Constructor argument properties

property DCTC.free_symbol: MatrixSymbol | MatrixElement | Symbol | str

The representation of the algebraic symbol to be used as the free parameter in the case where the CV map has a multiplicity of fixed points.

11.2.1.2.2 Read-only properties

property DCTC.matrix: MutableDenseMatrix

The matrix representation of the total D-CTC chronology-respecting (CR) output state prior to any post-processing.

11.2.1.2.3 Methods

```
DCTC.output_violating(
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
    simplify: bool | None = None,
    conjugate: bool | None = None,
    free_symbol: MatrixSymbol | MatrixElement | Symbol | str | None = None,
) → MutableDenseMatrix
```

Compute the matrix representation of the D-CTC chronology-violating (CV) state(s).

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the state. Defaults to `False`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ **free_symbol** (`str`) – The string representation of the algebraic symbol to be used as the free parameter in the case where the CV map has a multiplicity of fixed points. Defaults to the value of `self.free_symbol`.

Returns

`mat` – The matrix representation of the CV output state.

```
DCTC.output_respecting(
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
    simplify: bool | None = None,
    conjugate: bool | None = None,
    postprocess: bool | None = None,
    free_symbol: MatrixSymbol | MatrixElement | Symbol | str | None = None,
) → MutableDenseMatrix
```

Compute the matrix representation of the D-CTC chronology-respecting (CR) state(s) (including any post-processing).

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the state. Defaults to `False`.

- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ **postprocess** (`bool`) – Whether to post-process the state (i.e., perform the circuit's traces and postselections). Defaults to `True`.
- ▶ **free_symbol** (`str`) – The string representation of the algebraic symbol to be used as the free parameter in the case where the CV map has a multiplicity of fixed points. Defaults to the value of `self.free_symbol`.

Returns

`mat` – The matrix representation of the (post-processed) CR output state.

DCTC.output()

```
conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
simplify: bool | None = None,
conjugate: bool | None = None,
postprocess: bool | None = None,
free_symbol: MatrixSymbol | MatrixElement | Symbol | str | None = None,
) → MutableDenseMatrix
```

An alias for the `output_respecting()` (page 165) method.

Useful for polymorphism.

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the state. Defaults to `False`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ **postprocess** (`bool`) – Whether to post-process the state (i.e., perform the circuit's traces and postselections). Defaults to `True`.
- ▶ **free_symbol** (`str`) – The string representation of the algebraic symbol to be used as the free parameter in the case where the CV map has a multiplicity of fixed points. Defaults to the value of `self.free_symbol`.

Returns

`mat` – The matrix representation of the (post-processed) CR output state.

DCTC.state_violating()

```
conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
simplify: bool | None = None,
conjugate: bool | None = None,
norm: bool | Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str | None = None,
label: str | None = None,
notation: str | None = None,
traces: list[int] | None = None,
debug: bool | None = None,
free_symbol: MatrixSymbol | MatrixElement | Symbol | str | None = None,
) → QuantumState (page 83)
```

Compute the D-CTC chronology-violating (CV) state(s) as a `QuantumState` (page 83) instance.

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the state before committing it to the `matrix` property. Defaults to `False`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ **norm** (`bool | num | sym | str`) – The value to which the state is normalized. If `True`, normalizes to a value of 1. If `False`, does not normalize. Defaults to `False`.
- ▶ **label** (`str`) – The unformatted string used to represent the state in mathematical expressions. Must have a non-zero length. Defaults to " `ρ` " (if `form == "matrix"`) or " `ψ` " (if `form == "vector"`).
- ▶ **notation** (`str`) – The formatted string used to represent the state in mathematical expressions. When not `None`, overrides the value passed to `label`. Must have a non-zero length. Not intended to be set by the user in most cases. Defaults to `None`.
- ▶ **traces** (`list[int]`) – A list of indices of the CV systems (relative to the entire circuit) on which to perform partial traces. Defaults to `[]`.
- ▶ **free_symbol** (`str`) – The string representation of the algebraic symbol to be used as the free parameter in the case where the CV map has a multiplicity of fixed points. Defaults to the value of `self.free_symbol`.
- ▶ **debug** (`bool`) – Whether to print the internal state (held in `matrix`) on change. Defaults to `False`.

Returns

`QuantumState` – The CV output state as a `QuantumState` (page 83) instance.

```
DCTC.state_respecting(
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
    simplify: bool | None = None,
    conjugate: bool | None = None,
    norm: bool | Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str | None = None,
    label: str | None = None,
    notation: str | None = None,
    traces: list[int] | None = None,
    postprocess: bool | None = None,
    debug: bool | None = None,
    free_symbol: MatrixSymbol | MatrixElement | Symbol | str | None = None,
) → QuantumState (page 83)
```

Compute the D-CTC chronology-respecting (CR) state(s) as a `QuantumState` (page 83) instance.

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the state before committing it to the `matrix` property. Defaults to `False`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ **norm** (`bool | num | sym | str`) – The value to which the state is normalized. If `True`, normalizes to a value of 1. If `False`, does not normalize. Defaults to `False`.

- ▶ **label** (*str*) – The unformatted string used to represent the state in mathematical expressions. Must have a non-zero length. Defaults to " ρ " (if **form** == "matrix") or " ψ " (if **form** == "vector").
- ▶ **notation** (*str*) – The formatted string used to represent the state in mathematical expressions. When not **None**, overrides the value passed to **label**. Must have a non-zero length. Not intended to be set by the user in most cases. Defaults to **None**.
- ▶ **traces** (*list[int]*) – A list of indices of the CR systems (relative to the entire circuit) on which to perform partial traces. Performed regardless of the value of **postprocess**. Defaults to **[]**.
- ▶ **postprocess** (*bool*) – Whether to post-process the state (i.e., perform the circuit's traces and postselections). Defaults to **True**.
- ▶ **free_symbol** (*str*) – The string representation of the algebraic symbol to be used as the free parameter in the case where the CV map has a multiplicity of fixed points. Defaults to the value of **self.free_symbol**.
- ▶ **debug** (*bool*) – Whether to print the internal state (held in **matrix**) on change. Defaults to **False**.

Returns

QuantumState – The (post-processed) CR output state as a *QuantumState* (page 83) instance.

11.2.2 Postselected teleportation (P-CTCs)

11.2.2.1 Functions

```
pctc_violating(  
    input_respecting: MutableDenseMatrix | QuantumState (page 83),  
    gate: MutableDenseMatrix | QuantumGate (page 105),  
    systems_respecting: list[int],  
    systems_violating: list[int],  
) → MutableDenseMatrix
```

Calculate the chronology-violating (CV) state according to the P-CTC weak-measurement tomography expression for the prescription's CV map

$$\mathbf{p}_{\hat{U}}[\hat{\rho}] = \text{tr}_{\text{CR}}[\hat{U}(\hat{\rho} \otimes \frac{1}{d}\hat{I})\hat{U}^\dagger] \quad (11.3)$$

given the chronology-respecting (CR) input state **input_respecting** ($\hat{\rho}$) and (unitary) interaction described by **gate** (\hat{U}). Here, d is the dimensionality of the CV system's Hilbert space (assumed to be equivalent to that of its CR counterpart), while \hat{I} is the $d \times d$ identity matrix.

Parameters

- ▶ **input_respecting** (*mat* | *QuantumState* (page 83)) – The matrix representation of the chronology-respecting (CR) input state.
- ▶ **gate** (*mat* | *QuantumGate* (page 105)) – The matrix representation of the gate describing the (unitary) interaction between the CR and CV systems.
- ▶ **systems_respecting** (*list[int]*) – The numerical indices of the chronology-respecting (CR) subsystems.
- ▶ **systems_violating** (*list[int]*) – The numerical indices of the chronology-violating (CV) subsystems.

Returns

mat – The weak-measurement tomography expression for the P-CTC's CV state.

Note:

The validity of the expression used in this function to compute the P-CTC CV state for *non-qubit* systems has not been proven.

pctc_respecting(

```
    input_respecting: MutableDenseMatrix | QuantumState (page 83),
    gate: MutableDenseMatrix | QuantumGate (page 105),
    systems_respecting: list[int],
    systems_violating: list[int],
) → MutableDenseMatrix
```

Calculate the (non-renormalized) chronology-respecting (CR) state according to the P-CTC prescription's non-renormalizing CR map

$$\mathbf{P}_{\hat{U}}[\hat{\rho}] \propto \hat{\mathcal{P}}\hat{\rho}\hat{\mathcal{P}}^\dagger \quad (11.4)$$

given the chronology-respecting (CR) input state `input_respecting` ($\hat{\rho}$) and (unitary) interaction described by `gate` (\hat{U}). Here,

$$\hat{\mathcal{P}} \equiv \text{tr}_{\text{CV}}[\hat{U}] \quad (11.5)$$

is the P-CTC operator.

Note:

This function does not renormalize the output state as per the renormalized P-CTC map

$$\mathbf{P}_{\hat{U}}[\hat{\rho}] = \frac{\hat{\mathcal{P}}\hat{\rho}\hat{\mathcal{P}}^\dagger}{\text{tr}[\hat{\mathcal{P}}\hat{\rho}\hat{\mathcal{P}}^\dagger]} \quad (11.6)$$

Parameters

- ▶ `input_respecting` (`mat` | `QuantumState` (page 83)) – The matrix representation of the chronology-respecting (CR) input state.
- ▶ `gate` (`mat` | `QuantumGate` (page 105)) – The matrix representation of the gate describing the (unitary) interaction between the CR and CV systems.
- ▶ `systems_respecting` (`list[int]`) – The numerical indices of the chronology-respecting (CR) subsystems.
- ▶ `systems_violating` (`list[int]`) – The numerical indices of the chronology-violating (CV) subsystems.

Returns

`mat` – The solution of the P-CTC CR map.

11.2.2.2 Class

```
class PCTC(
    *args,
    **kwargs,
)
Bases: QuantumCTC (page 157)
```

A subclass for creating closed timelike curves described by the postselected teleportation prescription (P-CTCs) of quantum time travel.

This is built upon the [QuantumCTC](#) (page 157) class, and so inherits all of its attributes, properties, and methods.

Parameters

- ▶ ***args** – Variable-length argument list, passed directly to the constructor `__init__` of the superclass [QuantumGate](#) (page 105).
- ▶ ****kwargs** – Arbitrary keyword arguments, passed directly to the constructor `__init__` of the superclass [QuantumGate](#) (page 105).

Examples

For usage examples, please see the superclass [QuantumCTC](#) (page 157).

11.2.2.2.1 Read-only properties

`property PCTC.matrix: MutableDenseMatrix`

The matrix representation of the total P-CTC CR output state prior to any post-processing.

11.2.2.2.2 Methods

`PCTC.output_violating(`
 `conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,`
 `simplify: bool | None = None,`
 `conjugate: bool | None = None,`
`) → MutableDenseMatrix`

Compute the matrix representation of the P-CTC chronology-violating (CV) state.

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the state. Defaults to `False`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.

Returns

`mat` – The matrix representation of the CV output state.

Note:

The validity of the expression used in this method to compute the P-CTC CV state for *non-qubit* systems has not been proven.

`PCTC.output_respecting(`
 `conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,`
 `simplify: bool | None = None,`
 `conjugate: bool | None = None,`

```
postprocess: bool | None = None,
) → MutableDenseMatrix
```

Compute the matrix representation of the P-CTC chronology-respecting (CR) state (including any post-processing).

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the state. Defaults to `False`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ **postprocess** (`bool`) – Whether to post-process the state (i.e., perform the circuit's traces and postselections). Defaults to `True`.

Returns

`mat` – The matrix representation of the (post-processed) CR output state.

Note:

The output state is not renormalized.

```
PCTC.output(
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
```

```
    simplify: bool | None = None,
    conjugate: bool | None = None,
    postprocess: bool | None = None,
) → MutableDenseMatrix
```

An alias for the `output_respecting()` (page 170) method.

Useful for polymorphism.

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the state. Defaults to `False`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ **postprocess** (`bool`) – Whether to post-process the state (i.e., perform the circuit's traces and postselections). Defaults to `True`.

Returns

`mat` – The matrix representation of the (post-processed) CR output state.

Note:

The output state is not renormalized.

```
PCTC.state_violating(
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]] | None = None,
```

```
    simplify: bool | None = None,
```

```
simplify: bool | None = None,
conjugate: bool | None = None,
norm: bool | Number | generic | Basic | MatrixSymbol | MatrixElement | Sym-
bol | str | None = None,
label: str | None = None,
notation: str | None = None,
traces: list[int] | None = None,
debug: bool | None = None,
) → QuantumState (page 83)
```

Compute the P-CTC chronology-violating (CV) state as a [QuantumState](#) (page 83) instance.

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the state before committing it in the `matrix` property. Defaults to `False`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ **norm** (`bool | num | sym | str`) – The value to which the state is normalized. If `True`, normalizes to a value of 1. If `False`, does not normalize. Defaults to `False`.
- ▶ **label** (`str`) – The unformatted string used to represent the state in mathematical expressions. Must have a non-zero length. Defaults to "`ρ`" (if `form == "matrix"`) or "`ψ`" (if `form == "vector"`).
- ▶ **notation** (`str`) – The formatted string used to represent the state in mathematical expressions. When not `None`, overrides the value passed to `label`. Must have a non-zero length. Not intended to be set by the user in most cases. Defaults to `None`.
- ▶ **traces** (`list[int]`) – A list of indices of the CV systems (relative to the entire circuit) on which to perform partial traces. Defaults to `[]`.
- ▶ **debug** (`bool`) – Whether to print the internal state (held in `matrix`) on change. Defaults to `False`.

Returns

`QuantumState` – The CV output state as a [QuantumState](#) (page 83) instance.

Note:

The validity of the expression used in this method to compute the P-CTC CV state for *non-qubit* systems has not been proven.

```
PCTC.state_respecting(
    conditions: list[tuple[Number | generic | Basic | MatrixSymbol | MatrixEle-
        ment | Symbol | str, Number | generic | Basic | MatrixSymbol | MatrixElement
        | Symbol | str]] | None = None,
    simplify: bool | None = None,
    conjugate: bool | None = None,
    norm: bool | Number | generic | Basic | MatrixSymbol | MatrixElement | Sym-
        bol | str | None = None,
    label: str | None = None,
    notation: str | None = None,
    traces: list[int] | None = None,
    postprocess: bool | None = None,
    debug: bool | None = None,
) → QuantumState (page 83)
```

Compute the P-CTC chronology-respecting (CR) state as a [QuantumState](#) (page 83) instance.

Parameters

- ▶ **conditions** (`list[tuple[num | sym | str, num | sym | str]]`) – Algebraic conditions to be applied to the state. Defaults to the value of `self.conditions`.
- ▶ **simplify** (`bool`) – Whether to perform algebraic simplification on the state before committing it to the `matrix` property. Defaults to `False`.
- ▶ **conjugate** (`bool`) – Whether to perform Hermitian conjugation on the state. Defaults to `False`.
- ▶ **norm** (`bool | num | sym | str`) – The value to which the state is normalized. If `True`, normalizes to a value of 1. If `False`, does not normalize. Defaults to `False`.
- ▶ **label** (`str`) – The unformatted string used to represent the state in mathematical expressions. Must have a non-zero length. Defaults to " ρ " (if `form == "matrix"`) or " ψ " (if `form == "vector"`).
- ▶ **notation** (`str`) – The formatted string used to represent the state in mathematical expressions. When not `None`, overrides the value passed to `label`. Must have a non-zero length. Not intended to be set by the user in most cases. Defaults to `None`.
- ▶ **traces** (`list[int]`) – A list of indices of the CR systems (relative to the entire circuit) on which to perform partial traces. Performed regardless of the value of `postprocess`. Defaults to `[]`.
- ▶ **postprocess** (`bool`) – Whether to post-process the state (i.e., perform the circuit's traces and postselections). Defaults to `True`.
- ▶ **debug** (`bool`) – Whether to print the internal state (held in `matrix`) on change. Defaults to `False`.

Returns

`QuantumState` – The (post-processed) CR output state as a [QuantumState](#) (page 83) instance.

Note:

The output state is not renormalized if `norm` is `False`.

This module provides a collection of core functions for constructing matrices in quantum mechanics.

```
from qchronology.mechanics.matrices import vector_basis, ket, bra, quantum_state, encode,
↪ decode_slow, decode, decode_fast, decode_multiple
```

12.1 Functions

```
vector_basis(
    dim: int,
) → list[MutableDenseMatrix]
```

Creates an ordered list of column vectors that form an orthonormal basis for a `dim`-dimensional Hilbert space.

Parameters

`dim` (`int`) – The dimensionality of the vector basis. Must be a non-negative integer.

Returns

`list[int]` – An ordered list of basis vectors.

Examples

```
>>> vector_basis(2)
[Matrix([
    [1],
    [0]]),
 Matrix([
    [0],
    [1]])]
```

```
>>> vector_basis(3)
[Matrix([
    [1],
    [0],
    [0]]),
 Matrix([
    [0],
    [1],
    [0]]),
 Matrix([
    [0],
    [0],
    [1]])]
```

```
ket(
    spec: int | list[int],
    dim: int | None = None,
) → MutableDenseMatrix
```

Creates a normalized ket (column) basis vector corresponding to the (multipartite) computational-basis value(s) of `spec` in a `dim`-dimensional Hilbert space.

In mathematical notation, `spec` describes the value of the ket vector, e.g., a `spec` of `[i,j,k]` corresponds to the ket vector $|i,j,k\rangle$ (for some non-negative integers `i`, `j`, and `k`).

Parameters

- ▶ `spec (int | list[int])` – A non-negative integer or a list of such types.
- ▶ `dim (int)` – The dimensionality of the vector. Must be a non-negative integer. Defaults to 2.

Returns

`mat` – A normalized column vector.

Examples

```
>>> ket(0)
Matrix([
[1],
[0]))
```

```
>>> ket(1)
Matrix([
[0],
[1]))
```

```
>>> ket([2, 1], dim=3)
Matrix([
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[1],
[0]))
```

```
bra(
    spec: int | list[int],
    dim: int | None = None,
) → MutableDenseMatrix
```

Creates a normalized bra (row) basis vector corresponding to the (multipartite) computational-basis value(s) of `spec` in a `dim`-dimensional dual Hilbert space.

In mathematical notation, `spec` describes the value of the bra vector, e.g., a `spec` of `[i,j,k]` corresponds to the bra vector $\langle i,j,k|$ (for some non-negative integers `i`, `j`, and `k`).

Parameters

- ▶ `spec (int | list[int])` – A non-negative integer or a list of such types.
- ▶ `dim (int)` – The dimensionality of the vector. Must be a non-negative integer. Defaults to 2.

Returns

`mat` – A normalized row vector.

Examples

```
>>> bra(0)
Matrix([[1, 0]])
```

```
>>> bra(1)
Matrix([[0, 1]])
```

```
>>> bra([0, 2], dim=3)
Matrix([[0, 0, 1, 0, 0, 0, 0, 0, 0]])
```

quantum_state(

spec: *MutableDenseMatrix* | *ndarray* | *list[list[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str]]* | *list[tuple[Number | generic | Basic | MatrixSymbol | MatrixElement | Symbol | str, int | list[int]]]*,

form: *str* | *None* = *None*,

kind: *str* | *None* = *None*,

dim: *int* | *None* = *None*,

) → *MutableDenseMatrix*

Constructs a *dim*-dimensional matrix or vector representation of a quantum state from a given specification *spec*.

Parameters

- ▶ ***spec*** – The specification of the quantum state. Provides a complete description of the state's values in a standard *dim*-dimensional basis. Can be one of:
 - a SymPy matrix (*mat*)
 - a NumPy array (*arr*)
 - a list of lists of numerical, symbolic, or string expressions that collectively specify a vector or (square) matrix (*list[list[num | sym | str]]*)
 - a list of 2-tuples of numerical, symbolic, or string coefficients paired their respective number-basis specification (*list[tuple[num | sym | str, int | list[int]]]*)
- ▶ ***form (str)*** – A string specifying the *form* for the quantum state to take. Can be either of "*vector*" or "*matrix*". Defaults to "*matrix*".
- ▶ ***kind (str)*** – A string specifying the *kind* for the quantum state to take. Can be either of "*mixed*" or "*pure*". Defaults to "*mixed*".
- ▶ ***dim (int)*** – The dimensionality of the quantum state's Hilbert space. Must be a non-negative integer. Defaults to **2**.

Returns

mat – The matrix or vector representation of the quantum state.

Examples

```
>>> quantum_state([('a', [0]), ('b', [1])], form="vector", kind="pure", dim=2)
Matrix([
[a],
[b]])
```

```
>>> quantum_state([('a', [0]), ('b', [1])], form="matrix", kind="pure", dim=2)
Matrix([
[a*conjugate(a), a*conjugate(b)],
[b*conjugate(a), b*conjugate(b)]])
```

```
>>> quantum_state([('a', [0]), ('b', [1])], form="matrix", kind="mixed", dim=2)
Matrix([

```

(continues on next page)

(continued from previous page)

```
[a, 0],  
[0, b])
```

```
>>> quantum_state(  
...     spec=[("a", [0]), ("b", [1]), ("c", [2])],  
...     form="vector",  
...     kind="pure",  
...     dim=3,  
... )  
Matrix([  
[a],  
[b],  
[c]])
```

```
>>> quantum_state(  
...     spec=[("a", [0, 0]), ("b", [1, 1])],  
...     form="vector",  
...     kind="pure",  
...     dim=2,  
... )  
Matrix([  
[a],  
[0],  
[0],  
[b]])
```

```
>>> quantum_state([[ "a", "b"], [ "c", "d"]], form="matrix", kind="mixed", dim=2)  
Matrix([  
[a, b],  
[c, d]])
```

```
>>> matrix = sp.Matrix([[ "a", "b"], [ "c", "d"]])  
>>> quantum_state(matrix, form="matrix", kind="mixed", dim=2)  
Matrix([  
[a, b],  
[c, d]])
```

encode
integer: int,
num_systems: int | None = None,
dim: int | None = None,
reverse: bool | None = None,
output_list: bool | None = None,
) → MutableDenseMatrix

Encodes a non-negative integer as a single quantum state vector (ket).

This is a kind of unsigned integer encoding. It creates a base-*dim* numeral system representation of *integer* as an (ordered) list of encoded digits. Returns this list if *output_list* is True, otherwise returns the corresponding ket vector (i.e., a ket vector with a spec of these digits).

Parameters

- ▶ **integer** (int) – The non-negative integer to be encoded.
- ▶ **num_systems** (int) – The number of systems (e.g., qubits) necessary to represent the integer in the encoding. Must be a non-negative integer. If **None**, it automatically increases to the

smallest possible number of systems with which the given `integer` can be encoded.

- ▶ `dim (int)` – The dimensionality (or base) of the encoding. Must be a non-negative integer. Defaults to `2`.
 - ▶ `reverse (str)` – Whether to reverse the ordering of the resulting encoded state.
 - If `reverse` is `False`, the significance of the digits *decreases* along the list (i.e., the least-significant digit is last).
 - If `reverse` is `True`, the significance of the digits *increases* along the list (i.e., the least-significant digit is first).
- Defaults to `False`.
- ▶ `output_list (bool)` – Whether to output a list of encoded digits instead of an encoded state. Defaults to `False`.

Returns

- ▶ `mat` – A normalized column vector (if `output_list` is `False`).
- ▶ `list[int]` – An ordered list of the encoded digits (if `output_list` is `True`).

Examples

```
>>> encode(3, num_systems=2)
Matrix([
[0],
[0],
[0],
[0],
[1]])
```

```
>>> encode(7, num_systems=2, dim=3)
Matrix([
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[1],
[0]]))
```

```
>>> encode(264, num_systems=3, dim=10, output_list=True)
[2, 6, 4]
```

```
>>> encode(115, num_systems=8, output_list=True)
[0, 1, 1, 1, 0, 0, 1, 1]
```

```
>>> encode(115, num_systems=8, output_list=True, reverse=True)
[1, 1, 0, 0, 1, 1, 1, 0]
```

```
decode_slow(
    matrix: mat | QuantumObject,
    dim: int | None = None,
    reverse: bool | None = None,
) → int
```

Decodes a quantum matrix or vector state to an unsigned integer.

Note:

The current method by which this particular implementation operates is accurate but slow. For a faster algorithm, use the `decode_fast()` function.

Note:

This function can also be called using the alias `decode()`.

Parameters

- ▶ `matrix` (`mat | QuantumObject`) – The quantum (matrix or vector) state to be decoded.
- ▶ `dim` (`int`) – The dimensionality (or base) of the encoding. Must be a non-negative integer. Defaults to `2`.
- ▶ `reverse` (`str`) – Whether to reverse the digit ordering of the encoded state prior to decoding.
 - If `reverse` is `False`, the significance of the digits should *decrease* along the list (i.e., the least-significant digit is last).
 - If `reverse` is `True`, the significance of the digits should *increase* along the list (i.e., the least-significant digit is first).

Defaults to `False`.

Returns

`int` – The decoded (unsigned) integer.

Examples

```
>>> decode_slow(encode(64))
64
```

```
>>> matrix = sp.Matrix([0, 0, 0, 0, 1, 0, 0, 0])
>>> decode_slow(matrix)
4
```

decode(

```
    matrix: mat | QuantumObject,
    dim: int | None = None,
    reverse: bool | None = None,
) → int
```

An alias for the `decode_slow()` (page 179) function.

decode_fast(

```
    matrix: mat | QuantumObject,
    dim: int | None = None,
) → int
```

Decodes a quantum matrix or vector state to an unsigned integer.

Note:

The current method by which this particular implementation operates is fast but may be inaccurate (due to some computational shortcuts that may not work in all cases). For a slower but accurate algorithm, use the `decode_slow()` function.

Note:

The output cannot be reversed like in `decode_slow()`.

Parameters

- ▶ `matrix` (`mat | QuantumObject`) – The quantum (matrix or vector) state to be decoded.
- ▶ `dim` (`int`) – The dimensionality (or base) of the encoding. Must be a non-negative integer. Defaults to `2`.

Returns

`int` – The decoded (unsigned) integer.

Examples

```
>>> decode_fast(encode(2048))
2048
```

```
>>> matrix = sp.Matrix([0, 0, 1, 0, 0, 0, 0])
>>> decode_fast(matrix, dim=3)
2
```

```
decode_multiple(
    matrix: mat | QuantumObject,
    dim: int | None = None,
    reverse: bool | None = None,
) → list[tuple[int, num | sym]]
```

Decodes a quantum matrix or vector state to one or more unsigned integers with their respective probabilities.

Parameters

- ▶ `matrix` (`mat | QuantumObject`) – The quantum (matrix or vector) state to be decoded.
- ▶ `dim` (`int`) – The dimensionality (or base) of the encoding. Must be a non-negative integer. Defaults to `2`.
- ▶ `reverse` (`str`) – Whether to reverse the digit ordering of the encoded state prior to decoding.
 - If `reverse` is `False`, the significance of the digits should *decrease* along the list (i.e., the least-significant digit is last).
 - If `reverse` is `True`, the significance of the digits should *increase* along the list (i.e., the least-significant digit is first).

Defaults to `False`.

Returns

`list[tuple[int, num / sym]]` – The list of tuples of pairs of decoded (unsigned) integers and their corresponding probabilities.

Examples

```
>>> a, b = sp.symbols("a, b")
>>> matrix = a * encode(0) + b * encode(1)
>>> decode_multiple(matrix)
[(0, a*conjugate(a)), (1, b*conjugate(b))]
```

```
>>> matrix = sp.Matrix(["x", 0, 0, "y"])
>>> decode_multiple(matrix)
[(0, x*conjugate(x)), (3, y*conjugate(y))]
```

This module provides functions and a mixin for calculating quantum quantities.

```
from qhronology.mechanics.quantities import trace, purity, distance, fidelity, entropy,_
    mutual
from qhronology.mechanics.quantities import QuantitiesMixin
```

13.1 Functions

trace
`matrix: mat | QuantumObject,`
`) → num | sym`

Calculate the (complete) trace $\text{tr}[\hat{\rho}]$ of `matrix` ($\hat{\rho}$).

Parameters

`matrix (mat | QuantumObject)` – The input matrix.

Returns

`num / sym` – The trace of the input `matrix`.

Examples

```
>>> matrix = sp.Matrix([[["a", "b"], ["c", "d"]])
>>> trace(matrix)
a + d
```

```
>>> matrix = sp.MatrixSymbol("U", 3, 3).asMutable()
>>> trace(matrix)
U[0, 0] + U[1, 1] + U[2, 2]
```

purity
`state: mat | QuantumObject,`
`) → num | sym`

Calculate the purity (γ) of `state` ($\hat{\rho}$):

$$\gamma(\hat{\rho}) = \text{tr}[\hat{\rho}^2]. \quad (13.1)$$

Parameters

`state (mat | QuantumObject)` – The matrix representation of the input state.

Returns

`num / sym` – The purity of the input `state`.

Examples

```
>>> matrix = sp.Matrix([[["a", "b"], ["c", "d"]])
>>> purity(matrix)
a**2 + 2*b*c + d**2
```

```
>>> matrix = sp.Matrix(
...     [[ "a*conjugate(a)", "a*conjugate(b)" ], [ "b*conjugate(a)", "b*conjugate(b)" ]])
...
>>> purity(matrix)
(a*conjugate(a) + b*conjugate(b))**2
```

distance

`state_A: mat | QuantumObject,`
`state_B: mat | QuantumObject,`
`) → num | sym`

Calculate the trace distance (D) between two states `state_A` ($\hat{\rho}$) and `state_B` ($\hat{\tau}$):

$$D(\hat{\rho}, \hat{\tau}) = \frac{1}{2} \text{tr} |\hat{\rho} - \hat{\tau}|. \quad (13.2)$$

Parameters

- ▶ `state_A (mat | QuantumObject)` – The matrix representation of the first input state.
- ▶ `state_B (mat | QuantumObject)` – The matrix representation of the second input state.

Returns

`num / sym` – The trace distance between the inputs `state_A` and `state_B`.

Examples

```
>>> matrix_A = sp.Matrix([[["p", 0], [0, "1 - p"]]])
>>> matrix_B = sp.Matrix([[["q", 0], [0, "1 - q"]]])
>>> distance(matrix_A, matrix_B)
sqrt((p - q)*(conjugate(p) - conjugate(q)))
```

```
>>> matrix_A = sp.Matrix([[["1/sqrt(2)"], ["1/sqrt(2)"]]])
>>> matrix_B = sp.Matrix([[["1/sqrt(2)"], ["-1/sqrt(2)"]]])
>>> distance(matrix_A, matrix_B)
1
```

```
>>> matrix = sp.Matrix([[["a", "b"], ["c", "d"]]])
>>> distance(matrix, matrix)
0
```

fidelity

`state_A: mat | QuantumObject,`
`state_B: mat | QuantumObject,`
`) → num | sym`

Calculate the fidelity (F) between two states `state_A` ($\hat{\rho}$) and `state_B` ($\hat{\tau}$):

$$F(\hat{\rho}, \hat{\tau}) = \left(\text{tr} \sqrt{\sqrt{\hat{\rho}} \hat{\tau} \sqrt{\hat{\rho}}} \right)^2. \quad (13.3)$$

Parameters

- ▶ `state_A (mat | QuantumObject)` – The matrix representation of the first input state.
- ▶ `state_B (mat | QuantumObject)` – The matrix representation of the second input state.

Returns

`num / sym` – The fidelity between the inputs `state_A` and `state_B`.

Examples

```
>>> matrix_A = sp.Matrix([[ "a"], [ "b"]])
>>> matrix_B = sp.Matrix([[ "c"], [ "d"]])
>>> fidelity(matrix_A, matrix_A)
(a*conjugate(a) + b*conjugate(b))**2
>>> fidelity(matrix_B, matrix_B)
(c*conjugate(c) + d*conjugate(d))**2
>>> fidelity(matrix_A, matrix_B)
(a*conjugate(c) + b*conjugate(d))*(c*conjugate(a) + d*conjugate(b))
```

```
>>> matrix_A = sp.Matrix([[ "p", 0], [0, "1 - p"]])
>>> matrix_B = sp.Matrix([[ "q", 0], [0, "1 - q"]])
>>> fidelity(matrix_A, matrix_B)
(sqrt(p*q) + sqrt((1 - p)*(1 - q)))**2
```

```
>>> matrix_A = sp.Matrix([[ "1/sqrt(2)"], [ "1/sqrt(2)"]])
>>> matrix_B = sp.Matrix([[ "1/sqrt(2)"], [ "-1/sqrt(2)"]])
>>> fidelity(matrix_A, matrix_B)
0
```

entropy(
`state_A: mat | QuantumObject,`
`state_B: mat | QuantumObject | None = None,`
`base: num | None = None,`
`) → num | sym`

Calculate the relative von Neumann entropy (S) between two states `state_A` ($\hat{\rho}$) and `state_B` ($\hat{\tau}$):

$$S(\hat{\rho} \parallel \hat{\tau}) = \text{tr}[\hat{\rho}(\log_b \hat{\rho} - \log_b \hat{\tau})]. \quad (13.4)$$

If `state_B` is not specified (i.e., `None`), calculate the ordinary von Neumann entropy of `state_A` ($\hat{\rho}$) instead:

$$S(\hat{\rho}) = \text{tr}[\hat{\rho} \log_b \hat{\rho}]. \quad (13.5)$$

Here, b represents `base`, which is the dimensionality of the unit of information with which the entropy is measured.

Parameters

- ▶ `state_A` (`mat | QuantumObject`) – The matrix representation of the first input state.
- ▶ `state_B` (`mat | QuantumObject`) – The matrix representation of the second input state.
- ▶ `base` (`num`) – The dimensionality of the unit of information with which the entropy is measured. Defaults to `2`.

Returns

`num / sym` – The von Neumann entropy of the input `state_A` (if `state_B` is `None`) or the relative entropy between `state_A` and `state_B` (if `state_B` is not `None`).

Examples

```
>>> matrix = sp.Matrix([[ "a"], [ "b"]])
>>> entropy(matrix, base='d')
-(a*conjugate(a) + b*conjugate(b))**2*log(a*conjugate(a) + b*conjugate(b))/
```

```
>>> matrix_A = sp.Matrix([[["p", 0], [0, "1 - p"]]])
>>> matrix_B = sp.Matrix([[["q", 0], [0, "1 - q"]]])
>>> entropy(matrix_A, base="d")
(-p*log(p) + (p - 1)*log(1 - p))/log(d)
>>> entropy(matrix_B, base="d")
(-q*log(q) + (q - 1)*log(1 - q))/log(d)
>>> entropy(matrix_A, matrix_B, base="d")
(p*(log(p) - log(q)) - (p - 1)*(log(1 - p) - log(1 - q)))/log(d)
```

```
>>> matrix_A = sp.Matrix([[["1/sqrt(2)"], ["1/sqrt(2)"]]])
>>> matrix_B = sp.eye(2) / 2
>>> entropy(matrix_A)
0
>>> entropy(matrix_B)
1
>>> entropy(matrix_A, matrix_B)
1
>>> entropy(matrix_B, matrix_A)
-1
```

mutual(
 state: mat | QuantumObject,
 systems_A: int | list[int] | None = None,
 systems_B: int | list[int] | None = None,
 dim: int | None = None,
) → num | sym

Calculate the mutual information (I) between two subsystems `systems_A` (A) and `systems_B` (B) of a composite quantum system represented by `state` ($\rho^{A,B}$):

$$I(A : B) = S(\hat{\rho}^A) + S(\hat{\rho}^B) - S(\hat{\rho}^{A,B}) \quad (13.6)$$

where $S(\hat{\rho})$ is the von Neumann entropy of a state $\hat{\rho}$.

Parameters

- ▶ `state` (mat | QuantumObject) – The matrix representation of the composite input state.
- ▶ `systems_A` (int | list[int]) – The indices of the first subsystem. Defaults to `[0]`.
- ▶ `systems_B` (int | list[int]) – The indices of the second subsystem. Defaults to the complement of `systems_A` with respect to the entire composition of subsystems of `state`.
- ▶ `dim` (int) – The dimensionality of the composite quantum system (and its subsystems). Must be a non-negative integer. Defaults to `2`.

Returns

num / sym – The mutual information between the subsystems `systems_A` and `systems_B` of the composite input `state`.

Examples

```
>>> matrix = sp.Matrix([1, 0, 0, 1]) / sp.sqrt(2)
>>> mutual(matrix, [0], [1])
2
```

```
>>> matrix = sp.eye(4) / 4
>>> mutual(matrix, [0], [1])
0
```

13.2 Mixin

`class QuantitiesMixin`

A mixin for endowing classes with the ability to calculate various quantum quantities.

Any inheriting class must possess a matrix representation that can be accessed by either an `output()` method or a `matrix` property.

 **Note:**

The `QuantitiesMixin` (page 187) mixin is used exclusively by the `QuantumState` (page 83) class—please see the corresponding section (`Quantities` (page 98)) for documentation on its methods.

This module provides functions and a mixin for performing quantum operations.

```
from qhronology.mechanics.operations import densify, columnify, dagger, simplify, apply,
↪ rewrite, normalize, coefficient, partial_trace, measure, postselect
from qhronology.mechanics.operations import OperationsMixin
```

14.1 Functions

densify
 vector: mat | QuantumObject,
 \rightarrow mat

Convert `vector` to its corresponding matrix form via the outer product. If `vector` is a square matrix, it is unmodified.

Parameters

`vector` (mat) – The input vector.

Returns

`mat` – The outer product of `vector` with itself.

Examples

```
>>> vector = sp.Matrix([[["a"], ["b"]]])
>>> densify(vector)
Matrix([
[a*conjugate(a), a*conjugate(b)],
[b*conjugate(a), b*conjugate(b)]])
```

columnify
 vector: mat | QuantumObject,
 \rightarrow mat

Convert `vector` to its corresponding column vector form via transposition. If `vector` is a square matrix, it is unmodified.

Parameters

`vector` (mat) – The input vector.

Returns

`mat` – The column form of `vector`.

Examples

```
>>> vector = sp.Matrix([["a", "b"]])
>>> columnify(vector)
Matrix([
[a],
[b]])
```

```
dagger(  
    matrix: mat | QuantumObject,  
) → mat  
    Perform conjugate transposition on matrix.
```

Parameters

`matrix` (`mat`) – The input matrix.

Returns

`mat` – The conjugate transpose of `matrix`.

Examples

```
>>> matrix = sp.Matrix([[{"a"}, {"b}]])  
>>> dagger(matrix)  
Matrix([[conjugate(a), conjugate(b)]])
```

```
>>> matrix = sp.Matrix([[{"a", "b"}, {"c", "d"}]])  
>>> dagger(matrix)  
Matrix([  
[conjugate(a), conjugate(c)],  
[conjugate(b), conjugate(d)]])
```

```
simplify(  
    matrix: mat | QuantumObject,  
) → mat  
    Simplify matrix using a powerful (albeit slow) algorithm.
```

Parameters

`matrix` (`mat` | `QuantumObject`) – The matrix to be simplified.

Returns

`mat` – The simplified version of `matrix`.

Examples

```
>>> matrix = sp.Matrix(  
...     [  
...         [  
...             ["(a**2 - 1)/(a - 1) - 1", "log(cos(b) + I*sin(b))/I"],  
...             ["acos((exp(I*c) + exp(-I*c))/2)", "d**log(E*(sin(d)**2 + cos(d)**2))  
...         ],  
...     ]  
... )  
>>> simplify(matrix)  
Matrix([  
[a, b],  
[c, d]])
```

```
apply(  
    matrix: mat | QuantumObject,  
    function: Callable,  
    arguments: dict[str, Any] | None = None,  
) → mat
```

Apply a Python function (`function`) to `matrix`.

Useful when used with SymPy's symbolic-manipulation functions, such as:

- ▶ `apart()`
- ▶ `cancel()`
- ▶ `collect()`
- ▶ `expand()`
- ▶ `factor()`
- ▶ `simplify()`

More can be found at:

- ▶ SymPy documentation: Simplification²³
- ▶ SymPy documentation: Simplify²⁴

Parameters

- ▶ `matrix (mat | QuantumObject)` – The matrix to be transformed.
- ▶ `function (Callable)` – A Python function. Its first non-keyword argument must be able to take a mathematical expression or a matrix/array of such types.
- ▶ `arguments (dict[str, str])` – A dictionary of keyword arguments (both keys and values as strings) to pass to the `function` call. Defaults to `{}`.

Returns

`mat` – The transformed version of `matrix`.

Examples

```
>>> matrix = sp.Matrix([[{"x*y**2 - 2*x*y*z + x*z**2 + y**2 - 2*y*z + z**2}/(x**2 - 1)]])
>>> apply(matrix, function=sp.cancel)
Matrix([[((y**2 - 2*y*z + z**2)/(x - 1))])
>>> apply(matrix, function=sp.collect, arguments={"syms": "x"})
Matrix([[((x*(y**2 - 2*y*z + z**2) + y**2 - 2*y*z + z**2)/(x**2 - 1))])
>>> apply(matrix, function=sp.collect, arguments={"syms": "y"})
Matrix([[((x*z**2 + y**2*(x + 1) + y*(-2*x*z - 2*z) + z**2)/(x**2 - 1))])
>>> apply(matrix, function=sp.collect, arguments={"syms": "z"})
Matrix([[((x*y**2 + y**2 + z**2*(x + 1) + z*(-2*x*y - 2*y))/(x**2 - 1))])
>>> apply(matrix, function=sp.expand)
Matrix([[[(x*y**2/(x**2 - 1) - 2*x*x*y*z/(x**2 - 1) + x*x*z**2/(x**2 - 1) + y**2/(x**2 - 1) - 2*y*z/(x**2 - 1) + z**2/(x**2 - 1))]])
>>> apply(matrix, function=sp.factor)
Matrix([[((y - z)**2/(x - 1))]])
```

```
rewrite(
    matrix: mat | QuantumObject,
    function: Callable,
) → mat
```

Rewrite the elements of `matrix` using the given mathematical function (`function`).

Useful when used with SymPy's mathematical functions, such as:

- ▶ `exp()`
- ▶ `log()`
- ▶ `sin()`

²³ <https://docs.sympy.org/latest/tutorials/intro-tutorial/simplification.html>

²⁴ <https://docs.sympy.org/latest/modules/simplify/simplify.html>

► `cos()`

Parameters

- `matrix` (`mat` | `QuantumObject`) – The matrix to be transformed.
- `function` (`Callable`) – A SymPy mathematical function.

Returns

`mat` – The transformed version of `matrix`.

Examples

```
>>> matrix = sp.Matrix([[ "cos(x)" ], [ "sin(x)" ]])
>>> rewrite(matrix, function=sp.exp)
Matrix([
[ exp(I*x)/2 + exp(-I*x)/2],
[-I*(exp(I*x) - exp(-I*x))/2]])
```

```
normalize(
    matrix: mat | QuantumObject,
    norm: num | sym | str | None = None,
) → mat
```

Normalize `matrix` to the value specified (`norm`).

Parameters

- `matrix` (`mat` | `QuantumObject`) – The matrix to be normalized.
- `norm` (`num` | `sym` | `str`) – The value to which the matrix is normalized. Defaults to `1`.

Returns

`mat` – The normalized version of `matrix`.

Examples

```
>>> matrix = sp.Matrix([["a"], ["b"]])
>>> normalize(matrix, norm=1)
Matrix([
[a/sqrt(a*conjugate(a) + b*conjugate(b))],
[b/sqrt(a*conjugate(a) + b*conjugate(b))]])
```

```
>>> matrix = sp.Matrix([["a", "b"], ["c", "d"]])
>>> normalize(matrix, norm="n")
Matrix([
[a*n/(a + d), b*n/(a + d)],
[c*n/(a + d), d*n/(a + d)])
```

```
coefficient(
    matrix: mat | QuantumObject,
    scalar: num | sym | str | None = None,
) → mat
```

Multiply `matrix` by a scalar value (`scalar`).

Parameters

- `matrix` (`mat` | `QuantumObject`) – The matrix to be scaled.
- `scalar` (`num` | `sym` | `str`) – The value by which the state is multiplied. Defaults to `1`.

Returns

mat – The scaled version of `matrix`.

Examples

```
>>> matrix = sp.Matrix([[["1"], ["1"]]])
>>> coefficient(matrix, scalar=1 / sp.sqrt(2))
Matrix([
[sqrt(2)/2],
[sqrt(2)/2]])
```

```
>>> matrix = sp.Matrix([["a"], ["b"]])
>>> coefficient(matrix, scalar="exp(I*x)")
Matrix([
[a*exp(I*x)],
[b*exp(I*x)]])
```

partial_trace(

```
    matrix: mat | QuantumObject,
    targets: int | list[int] | None = None,
    discard: bool | None = None,
    dim: int | None = None,
    optimize: bool | None = None,
) → num | sym | mat
```

Compute and return the partial trace of a matrix.

Parameters

- ▶ **matrix** (*mat*) – The matrix on which to perform the partial trace operation.
- ▶ **targets** (*int* | *list[int]*) – The numerical index/indices of the subsystem(s) to be partially traced over. Defaults to `[]`.
- ▶ **discard** (*bool*) – Whether the systems corresponding to the indices given in **targets** are to be discarded (`True`) or kept (`False`). Defaults to `True`.
- ▶ **dim** (*int*) – The dimensionality of the matrix. Must be a non-negative integer. Defaults to `2`.
- ▶ **optimize** (*bool*) – Whether to optimize the implementation’s algorithm. Can greatly increase the computational efficiency at the cost of a larger memory footprint during computation. Defaults to `True`.

Returns

mat – The reduced matrix.

Examples

```
>>> matrix = sp.Matrix([["a"], ["b"], ["c"], ["d"]])
>>> partial_trace(matrix, targets=[0], dim=2)
Matrix([
[a*conjugate(a) + c*conjugate(c), a*conjugate(b) + c*conjugate(d)],
[b*conjugate(a) + d*conjugate(c), b*conjugate(b) + d*conjugate(d)]])
>>> partial_trace(matrix, targets=[1], dim=2)
Matrix([
[a*conjugate(a) + b*conjugate(b), a*conjugate(c) + b*conjugate(d)],
[c*conjugate(a) + d*conjugate(b), c*conjugate(c) + d*conjugate(d)]])
```

```
>>> matrix = sp.Matrix([[["a", 0, 0, 0], [0, "b", 0, 0], [0, 0, "c", 0], [0, 0, 0,
    ↪"d"]]])
>>> partial_trace(matrix, targets=[0], discard=True, dim=2)
Matrix([
[a + c, 0],
[0, b + d]])
>>> partial_trace(matrix, targets=[1], discard=True, dim=2)
Matrix([
[a + b, 0],
[0, c + d]])
```

measure(

```
matrix: mat | QuantumObject,
operators: list[mat | arr | QuantumObject],
targets: int | list[int],
observable: bool | None = None,
statistics: bool | None = None,
dim: int | None = None,
) → mat | list[num | sym]
```

Perform a quantum measurement on one or more systems (indicated in `targets`) of `matrix`.

This function has two main modes of operation:

- ▶ When `statistics` is `True`, the (reduced) state ($\hat{\rho}$) (residing on the systems indicated in `targets`) is measured and the set of resulting statistics is returned. This takes the form of an ordered list of values $\{p_i\}_i$ associated with each given operator, where:

- $p_i = \text{tr}[\hat{K}_i^\dagger \hat{K}_i \hat{\rho}]$ (measurement probabilities) when `observable` is `False` (`operators` is a list of Kraus operators or projectors \hat{K}_i)
- $p_i = \text{tr}[\hat{O}_i \hat{\rho}]$ (expectation values) when `observable` is `True` (`operators` is a list of observables \hat{O}_i)

- ▶ When `statistics` is `False`, the (reduced) state ($\hat{\rho}$) (residing on the systems indicated in `targets`) is measured and mutated it according to its predicted post-measurement form (i.e., the sum of all possible measurement outcomes). This yields the transformed states:

- When `observable` is `False`:

$$\hat{\rho}' = \sum_i \hat{K}_i \hat{\rho} \hat{K}_i^\dagger. \quad (14.1)$$

- When `observable` is `True`:

$$\hat{\rho}' = \sum_i \text{tr}[\hat{O}_i \hat{\rho}] \hat{O}_i. \quad (14.2)$$

In the case where `operators` contains only a single item (\hat{K}) and the current state ($|\psi\rangle$) is a vector form, the transformation of the state is in accordance with the rule

$$|\psi'\rangle = \frac{\hat{K}|\psi\rangle}{\sqrt{\langle\psi|\hat{K}^\dagger \hat{K}|\psi\rangle}} \quad (14.3)$$

when `observable` is `False`. In all other mutation cases, the post-measurement state is a matrix, even if the pre-measurement state was a vector.

The items in the list `operators` can also be vectors (e.g., $|\xi_i\rangle$), in which case each is converted into its corresponding operator matrix representation (e.g., $|\xi_i\rangle\langle\xi_i|$) prior to any measurements.

Parameters

- ▶ `matrix` (`mat | QuantumObject`) – The matrix to be measured.

- ▶ **operators** (`list[mat | arr | QuantumObject]`) – The operator(s) with which to perform the measurement. These would typically be a (complete) set of Kraus operators forming a POVM, a (complete) set of (orthogonal) projectors forming a PVM, or a set of observables constituting a complete basis for the relevant state space.
- ▶ **targets** (`int | list[int]`) – The numerical indices of the subsystem(s) to be measured. They must be consecutive, and their number must match the number of systems spanned by all given operators. Indexing begins at `0`. All other systems are discarded (traced over) in the course of performing the measurement.
- ▶ **observable** (`bool`) – Whether to treat the items in **operators** as observables instead of Kraus operators or projectors. Defaults to `False`.
- ▶ **statistics** (`bool`) – Whether to return a list of probabilities (`True`) or transform **matrix** into a post-measurement probabilistic sum of all outcomes (`False`). Defaults to `False`.
- ▶ **dim** (`int`) – The dimensionality of **matrix** and the item(s) of **operators**. Must be a non-negative integer. Defaults to `2`.

Returns

- ▶ **mat** – The post-measurement **matrix**. Returned only if **statistics** is `False`.
- ▶ **num / sym / list[num / sym]** – A list of probabilities corresponding to each operator given in **operators**. Returned only if **statistics** is `True`.

Note:

This method does not check for validity of supplied POVMs or the completeness of sets of observables, nor does it renormalize the post-measurement state.

Examples

```
>>> matrix = sp.Matrix([[["a"], ["b"]]])
>>> plus = sp.Matrix([[1 / sp.sqrt(2)], [1 / sp.sqrt(2)]])
>>> minus = sp.Matrix([[1 / sp.sqrt(2)], [-1 / sp.sqrt(2)]])
>>> measure(matrix, operators=[plus, minus], targets=[0], observable=False,
...           ↴statistics=True)
[a*conjugate(a)/2 + a*conjugate(b)/2 + b*conjugate(a)/2 + b*conjugate(b)/2,
 a*conjugate(a)/2 - a*conjugate(b)/2 - b*conjugate(a)/2 + b*conjugate(b)/2]
>>> measure(matrix, operators=[plus, minus], targets=[0], observable=False,
...           ↴statistics=False)
Matrix([
[a*conjugate(a)/2 + b*conjugate(b)/2, a*conjugate(b)/2 + b*conjugate(a)/2],
[a*conjugate(b)/2 + b*conjugate(a)/2, a*conjugate(a)/2 + b*conjugate(b)/2]])
```

```
>>> matrix = sp.Matrix([[["a"], ["b"]]])
>>> I = sp.Matrix([[1, 0], [0, 1]])
>>> X = sp.Matrix([[0, 1], [1, 0]])
>>> Y = sp.Matrix([[0, -sp.I], [sp.I, 0]])
>>> Z = sp.Matrix([[1, 0], [0, -1]])
>>> measure(matrix, operators=[I, X, Y, Z], targets=[0], observable=True,
...           ↴statistics=True)
[a*conjugate(a) + b*conjugate(b),
 a*conjugate(b) + b*conjugate(a),
 I*(a*conjugate(b) - b*conjugate(a)),
 a*conjugate(a) - b*conjugate(b)]
>>> measure(matrix, operators=[I, X, Y, Z], targets=[0], observable=True,
...           ↴statistics=False)
```

(continues on next page)

(continued from previous page)

```
Matrix([
    [2*a*conjugate(a), 2*a*conjugate(b)],
    [2*b*conjugate(a), 2*b*conjugate(b)]])
```

postselect(
matrix: mat | QuantumObject,
postselections: list[tuple[mat | arr | QuantumObject, int]],
dim: int | None = None,
 $\lambda \rightarrow \text{mat} | \text{list[num | sym]}$

Perform postselection on `matrix` against the operator(s) specified in `postselections`.

The postselections can be given in either vector or matrix form. For the former, the transformation of the vector $|\Psi\rangle$ follows the standard rule

$$|\Psi'\rangle = \langle\phi|\Psi\rangle \quad (14.4)$$

where $|\phi\rangle$ is the postselection vector. In the case of a matrix form $\hat{\omega}$, the notion of postselection of a matrix $\hat{\rho}$ naturally generalizes to

$$\hat{\rho}' = \text{tr}_{\{i\}}[\hat{\omega}\hat{\rho}] \quad (14.5)$$

where $\{i\}$ is the set of indices corresponding to the subsystem(s) upon which the postselection is performed.

If multiple postselections are supplied, `matrix` will be successively postselected in the order in which they are given. If a vector `matrix` is postselected against a matrix form, it will automatically be transformed into its matrix form via the outer product as necessary.

Parameters

- ▶ `matrix (mat | QuantumObject)` – The matrix to be measured.
- ▶ `postselections (list[tuple[mat | arr | QuantumObject, int]])` – A list of 2-tuples of vectors or matrix operators paired with the first (smallest) index of their postselection target systems.
- ▶ `dim (int)` – The dimensionality of `matrix` and the item(s) of `postselections`. Must be a non-negative integer. Defaults to `2`.

Returns

`mat` – The postselected form of `matrix`.

Examples

```
>>> matrix = sp.Matrix([[["a"]], [0], [0], ["b"]])
>>> zero = sp.Matrix([[1], [0]])
>>> one = sp.Matrix([[0], [1]])
>>> postselect(matrix, postselections=[(zero, [0])], dim=2)
Matrix([
[a],
[0]])
>>> postselect(matrix, postselections=[(one, [0])], dim=2)
Matrix([
[0],
[b]])
```

14.2 Mixin

`class OperationsMixin`

A mixin for endowing classes with the ability to have their `matrix` property mutated by various quantum operations.

 **Note:**

The `OperationsMixin` (page 197) mixin is used exclusively by the `QuantumState` (page 83) class—please see the corresponding section (*Operations* (page 91)) for documentation on its methods.

Part III

Examples

A small collection of examples that demonstrate the various uses and functionality of the package. It is divided into two chapters. The first is *quantum algorithms and protocols*, which consists of standard processes and circuits that are significant to the fields of quantum computing and quantum information science. The second chapter is *quantum closed timelike curves*, which contains more exotic circuits involving antichronological time travel, particularly in the presence of physically interesting unitary interactions.

15.1 Generation of a Bell state

15.1.1 Description

The circuit in [Figure 15.1](#) illustrates an algorithm for the generation of a Bell state ([3.54](#)) from primitive $|0\rangle$ states.

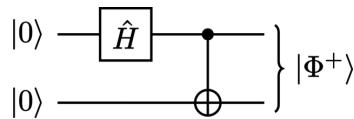


Figure 15.1: Generation of a Bell state.

The complete unitary transformation described by this circuit is the product

$$\hat{U} = C^0 \hat{X}^1 \cdot \hat{H}^0. \quad (15.1)$$

15.1.2 Implementation

[Listing 15.1: /text/examples/algorithms/generation_bell.py](#)

```
from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Hadamard, Not
from qhronology.quantum.circuits import QuantumCircuit

# Input
zero_state = VectorState(spec=[(1, [0])], label="0")

# Gates
HI = Hadamard(targets=[0], num_systems=2)
CN = Not(targets=[1], controls=[0], num_systems=2)

# Circuit
generator = QuantumCircuit(inputs=[zero_state, zero_state], gates=[HI, CN])
generator.diagram()

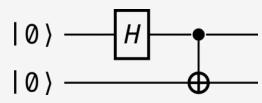
# Output
phi_plus = generator.state(label="Φ+")

# Results
phi_plus.print()
```

15.1.3 Output

15.1.3.1 Diagram

```
>>> generator.diagram()
```



15.1.3.2 State

```
>>> phi_plus.print()
|Φ+⟩ = sqrt(2)/2|0,0⟩ + sqrt(2)/2|1,1⟩
```

15.2 Generation of a GHZ state

15.2.1 Description

The circuit in Figure 15.2 illustrates an algorithm for the generation of a GHZ state (3.55) from primitive $|0\rangle$ states.

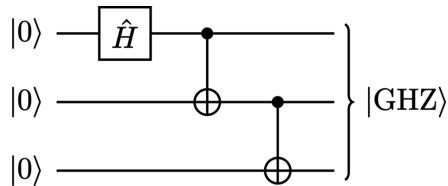


Figure 15.2: Generation of a GHZ state.

The complete unitary transformation described by this circuit is the product

$$\hat{U} = C^1 \hat{X}^2 \cdot C^0 \hat{X}^1 \cdot \hat{H}^0. \quad (15.2)$$

15.2.2 Implementation

Listing 15.2: /text/examples/algorithms/generation_ghz.py

```

from qchronology.quantum.states import VectorState
from qchronology.quantum.gates import Hadamard, Not
from qchronology.quantum.circuits import QuantumCircuit

# Input
zero_state = VectorState(spec=[(1, [0])], label="0")

# Gates
HII = Hadamard(targets=[0], num_systems=3)
CNI = Not(targets=[1], controls=[0], num_systems=3)
ICN = Not(targets=[2], controls=[1], num_systems=3)

# Circuit
generator = QuantumCircuit(
    inputs=[zero_state, zero_state, zero_state], gates=[HII, CNI, ICN])
generator.diagram()

# Output
ghz_state = generator.state(label="GHZ")

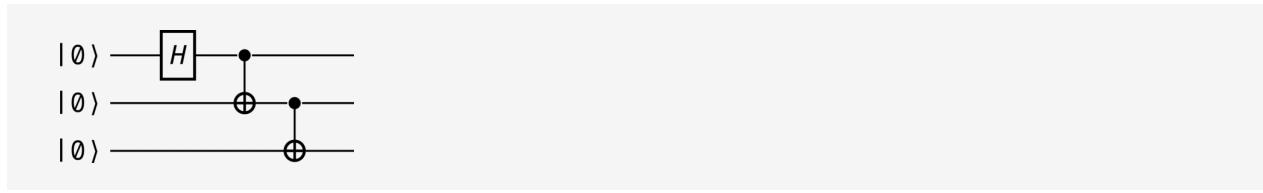
# Results
ghz_state.print()

```

15.2.3 Output

15.2.3.1 Diagram

```
>>> generator.diagram()
```



15.2.3.2 State

```
>>> ghz_state.print()
|GHZ> = sqrt(2)/2|0,0,0> + sqrt(2)/2|1,1,1>
```

15.3 Generation of a W state

15.3.1 Description

The circuit in Figure 15.3 illustrates an algorithm for the generation of a W state (3.56) from primitive $|0\rangle$ states.

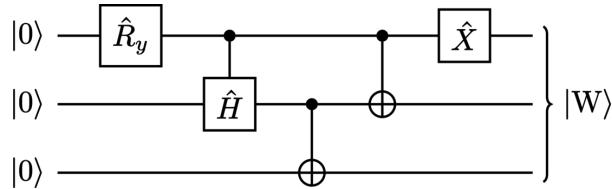


Figure 15.3: Generation of a W state.

The complete unitary transformation described by this circuit is the product

$$\hat{U} = \hat{X}^0 \cdot C^0 \hat{X}^1 \cdot C^1 \hat{X}^2 \cdot C^0 \hat{H}^1 \cdot \hat{R}_y^0(\theta) \quad (15.3)$$

where the y -rotation angle is $\theta = 2 \arccos\left(\frac{1}{\sqrt{3}}\right)$.

15.3.2 Implementation

Listing 15.3: /text/examples/algorithms/generation_w.py

```

from qchronology.quantum.states import VectorState
from qchronology.quantum.gates import Rotation, Hadamard, Not, Pauli
from qchronology.quantum.circuits import QuantumCircuit

# Input
zero_state = VectorState(spec=[(1, [0])], label="0")

# Gates
RII = Rotation(axis=2, angle="2*acos(1/sqrt(3))", targets=[0], num_systems=3, label="R")
CHI = Hadamard(targets=[1], controls=[0], num_systems=3)
ICN = Not(targets=[2], controls=[1], num_systems=3)
CNI = Not(targets=[1], controls=[0], num_systems=3)
XII = Pauli(index=1, targets=[0], num_systems=3)

# Circuit
generator = QuantumCircuit(
    inputs=[zero_state, zero_state, zero_state], gates=[RII, CHI, ICN, CNI, XII])
generator.diagram()

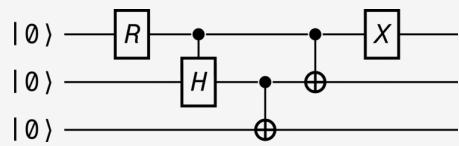
# Output
w_state = generator.state(label="W")

# Results
w_state.print()
  
```

15.3.3 Output

15.3.3.1 Diagram

```
>>> generator.diagram()
```



15.3.3.2 State

```
>>> w_state.print()
|w> = sqrt(3)/3|0,0,1> + sqrt(3)/3|0,1,0> + sqrt(3)/3|1,0,0>
```

15.4 CNOT (controlled-NOT)

15.4.1 Description

A CNOT (controlled-NOT) gate is a useful gate for performing a (qu)bit-wise sum of the state values of two quantum systems. For instance, given the states $|x\rangle$ and $|y\rangle$ in a d -dimensional basis $\{|n\rangle\}_{n=0}^{d-1}$, a CNOT operation has the action

$$|x\rangle \otimes |y\rangle \rightarrow |x\rangle \otimes |x \oplus y\rangle. \quad (15.4)$$

A circuit diagram of a CNOT gate appears in [Figure 15.4](#).

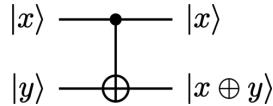


Figure 15.4: A CNOT (controlled-NOT) gate.

As the input states are both vectors, the application of the CNOT gate

$$\begin{aligned}\hat{U} &= C^0 \hat{X}^1 \\ &= \hat{I} \otimes \hat{I} + |1\rangle\langle 1| \otimes (\hat{\sigma}_x - \hat{I}),\end{aligned} \quad (15.5)$$

on the bipartite input $|x\rangle \otimes |y\rangle$ simply produces a vector state:

$$\begin{aligned}\mathbf{E}_{\hat{U}}[|x\rangle \otimes |y\rangle] &= C^0 \hat{X}^1 |x\rangle \otimes |y\rangle \\ &= |x\rangle \otimes |x \oplus y\rangle\end{aligned} \quad (15.6)$$

Observe how the value of the second system is now the sum of the inputs values (modulo the dimensionality), so if we simply discard (trace over) the first system, we obtain our desired result. A truth table for qubits appears in [Table 15.1](#).

Table 15.1: Truth table of the CNOT gate.

x	y	$x \oplus y$
0	0	0
1	0	1
0	1	1
1	1	0

15.4.2 Implementation

Listing 15.4: /text/examples/algorithms/cnot.py

```

from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Not
from qhronology.quantum.circuits import QuantumCircuit

# Input
first_state = VectorState(
    spec=[("a", [0]), ("b", [1])],
    symbols={"a": {"complex": True}, "b": {"complex": True}},
    conditions=[("a*conjugate(a) + b*conjugate(b)", "1")],
    label="x",
)
second_state = VectorState(

```

(continues on next page)

(continued from previous page)

```

spec=[("c", [0]), ("d", [1])],
symbols={"c": {"complex": True}, "d": {"complex": True}},
conditions=[("c*conjugate(c) + d*conjugate(d)", "1")],
label="y",
)

# Gate
CN = Not(targets=[1], controls=[0], num_systems=2)

# Circuit
circuit = QuantumCircuit(inputs=[first_state, second_state], gates=[CN])
circuit.diagram()

# Output
output_state = circuit.state(label="x, x ⊕ y")

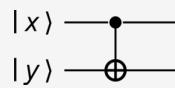
# Results
first_state.print()
second_state.print()
output_state.print()

```

15.4.3 Output

15.4.3.1 Diagram

```
>>> circuit.diagram()
```



15.4.3.2 States

```
>>> first_state.print()
|x⟩ = a|0⟩ + b|1⟩
```

```
>>> second_state.print()
|y⟩ = c|0⟩ + d|1⟩
```

```
>>> output_state.print()
|x, x ⊕ y⟩ = a*c|0,0⟩ + a*d|0,1⟩ + b*d|1,0⟩ + b*c|1,1⟩
```

15.5 CCNOT (controlled-controlled-NOT)

15.5.1 Description

A CCNOT (controlled-controlled-NOT) gate, also known as a *Toffoli gate*, is a simple extension to a CNOT gate (see *Example: CNOT* (page 207)). It is useful as a method of multiplying the values of two qubits (e.g., $|x\rangle$ and $|y\rangle$), and imprinting this result onto a third qubit (e.g., $|z\rangle$), e.g.,

$$|x\rangle \otimes |y\rangle \otimes |z\rangle \rightarrow |x\rangle \otimes |y\rangle \otimes |z \oplus xy\rangle. \quad (15.7)$$

Figure 15.5 visualizes this operation.

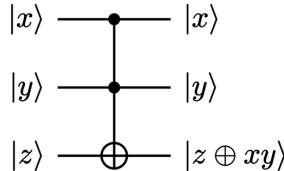


Figure 15.5: A CCNOT (controlled-controlled-NOT) gate.

As the input states are both vectors, the action of the (linear) CCNOT gate, described by the unitary

$$\begin{aligned} \hat{U} &= C^{0,1}\hat{X}^2 \\ &= \hat{I} \otimes \hat{I} + |1\rangle\langle 1| \otimes |1\rangle\langle 1| \otimes (\hat{\sigma}_x - \hat{I}), \end{aligned} \quad (15.8)$$

on the tripartite input $|x\rangle \otimes |y\rangle \otimes |z\rangle$ yields a vector state:

$$\begin{aligned} \mathbf{E}_{\hat{U}} [|x\rangle \otimes |y\rangle \oplus |z\rangle] &= C^{0,1}\hat{X}^2|x\rangle \otimes |y\rangle \oplus |z\rangle \\ &= |x\rangle \otimes |y\rangle \otimes |z \oplus xy\rangle \end{aligned} \quad (15.9)$$

Table 15.2 is a truth table for this operation in the context of qubits.

Table 15.2: Truth table of the CCNOT gate.

x	y	z	$z \oplus xy$
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	0

15.5.2 Implementation

Listing 15.5: /text/examples/algorithms/ccnot.py

```

from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Not
from qhronology.quantum.circuits import QuantumCircuit

# Input
first_state = VectorState(
    spec=[("a", [0]), ("b", [1])],
    symbols={"a": {"complex": True}, "b": {"complex": True}}},
  
```

(continues on next page)

(continued from previous page)

```

    conditions=[("a*conjugate(a) + b*conjugate(b)", "1")],
    label="x",
)
second_state = VectorState(
    spec=[("c", [0]), ("d", [1])],
    symbols={"c": {"complex": True}, "d": {"complex": True}},
    conditions=[("c*conjugate(c) + d*conjugate(d)", "1")],
    label="y",
)
third_state = VectorState(
    spec=[("u", [0]), ("v", [1])],
    symbols={"u": {"complex": True}, "v": {"complex": True}},
    conditions=[("u*conjugate(u) + v*conjugate(v)", "1")],
    label="z",
)

# Gate
CCN = Not(targets=[2], controls=[0, 1], num_systems=3)

# Circuit
circuit = QuantumCircuit(inputs=[first_state, second_state, third_state], gates=[CCN])
circuit.diagram()

# Output
output_state = circuit.state(label="x, y, z ⊕ xy")

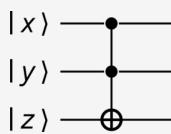
# Results
first_state.print()
second_state.print()
third_state.print()
output_state.print()

```

15.5.3 Output

15.5.3.1 Diagram

```
>>> circuit.diagram()
```



15.5.3.2 States

```
>>> first_state.print()
|x⟩ = a|0⟩ + b|1⟩
```

```
>>> second_state.print()
|y⟩ = c|0⟩ + d|1⟩
```

```
>>> third_state.print()
|z⟩ = u|0⟩ + v|1⟩
```

```
>>> output_state.print()
|x, y, z ⊕ xy⟩ = a*c*u|0,0,0⟩ + a*c*v|0,0,1⟩ + a*d*u|0,1,0⟩ + a*d*v|0,1,1⟩ + b*c*u|1,0,
↪0⟩ + b*c*v|1,0,1⟩ + b*d*v|1,1,0⟩ + b*d*u|1,1,1⟩
```

15.6 Toffoli decomposition

15.6.1 Description

An important research subfield of quantum computing involves the (de)construction of complex gates using sets of more primitive gates. This is termed *decomposition*, and mostly concerns the task of finding the (most) minimal *gate set* (i.e., most restricted) for any given gate. It is an interesting problem as primitive gates are in fact usually the only gates which can be directly implemented in many physical quantum computers. Consequently, finding compositions of such primitives that correctly reconstruct more complex gates is necessary in the pursuit of executing more advanced algorithms. Balancing the *depth* of the circuit (i.e., the longest path along its wires from input to output) with minimizing the size of the gate set also forms an avenue of research, as a circuit's depth generally correlates with its execution time (discounting any execution time differences between the various types of gates).

The Toffoli gate, also known as the CCNOT gate (see *Example: CCNOT* (page 209)), is one such gate that can be decomposed into more primitive gates. As it is an *entangling* gate, the Toffoli's gate set must include more than just the single-qubit rotation gates (such as the \hat{T} gate), as all such gates describe *non-entangling* operations. Thus, one such set of gates consists of the Hadamard gate \hat{H} and the CNOT gate, in addition to the \hat{T} gate (fourth root of \hat{Z} , i.e., $\hat{Z}^{\frac{1}{4}}$). Using this restricted set, one possible decomposition of the Toffoli gate appears in [Figure 15.6](#). Note that the conjugate transpose of any gate in the set, e.g., \hat{T}^\dagger , is considered to also be in the set.

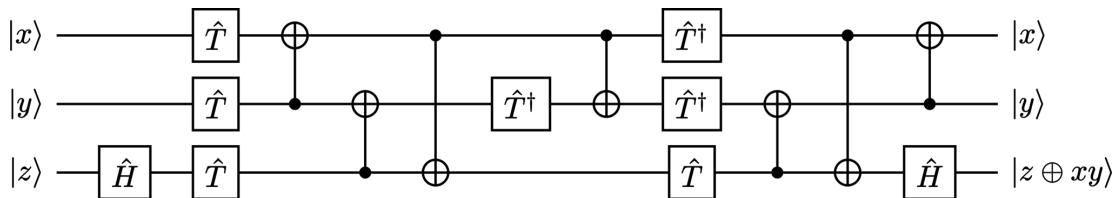


Figure 15.6: One possible decomposition of the Toffoli gate.

15.6.2 Implementation

Listing 15.6: /text/examples/algorithms/toffoli.py

```
from qhrontology.quantum.states import VectorState
from qhrontology.quantum.gates import Hadamard, Phase, Not, GateInterleave
from qhrontology.quantum.circuits import QuantumCircuit

import sympy as sp

# Input
first_state = VectorState(
    spec=[("a", [0]), ("b", [1])],
    symbols={"a": {"complex": True}, "b": {"complex": True}},
    conditions=[("a*conjugate(a) + b*conjugate(b)", "1")],
    label="x",
)
second_state = VectorState(
    spec=[("c", [0]), ("d", [1])],
    symbols={"c": {"complex": True}, "d": {"complex": True}},
    conditions=[("c*conjugate(c) + d*conjugate(d)", "1")],
    label="y",
)
third_state = VectorState(
    spec=[("u", [0]), ("v", [1])],
    symbols={"u": {"complex": True}, "v": {"complex": True}},
    conditions=[("u*conjugate(u) + v*conjugate(v)", "1")],
    label="z",
)
```

(continues on next page)

(continued from previous page)

```

        conditions=[("u*conjugate(u) + v*conjugate(v)", "1")],
        label="z",
)

# Gates
IIH = Hadamard(targets=[2], num_systems=3)
TII = Phase(exponent=sp.Rational(1, 4), targets=[0], num_systems=3, label="T")
ITI = Phase(exponent=sp.Rational(1, 4), targets=[1], num_systems=3, label="T")
IIT = Phase(exponent=sp.Rational(1, 4), targets=[2], num_systems=3, label="T")
TTT = GateInterleave(TII, ITI, IIT)
NCI = Not(targets=[0], controls=[1], num_systems=3)
INC = Not(targets=[1], controls=[2], num_systems=3)
CIN = Not(targets=[2], controls=[0], num_systems=3)
CNI = Not(targets=[1], controls=[0], num_systems=3)
ItI = Phase(
    exponent=sp.Rational(1, 4), conjugate=True, targets=[1], num_systems=3, label="T^"
)
tII = Phase(
    exponent=sp.Rational(1, 4), conjugate=True, targets=[0], num_systems=3, label="T^†"
)
ttT = GateInterleave(tII, ItI, IIT)
NCH = GateInterleave(NCI, IIH)

# Circuit
circuit = QuantumCircuit(
    inputs=[first_state, second_state, third_state],
    gates=[IIH, TTT, NCI, INC, CIN, ItI, CNI, ttT, INC, CIN, NCH],
)
circuit.diagram(force_separation=True)

# Output
output_state = circuit.state(label="x, y, z ⊕ xy")

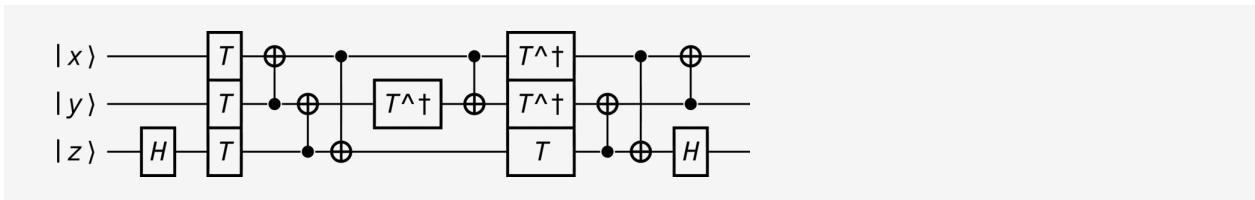
# Results
first_state.print()
second_state.print()
third_state.print()
output_state.print()

```

15.6.3 Output

15.6.3.1 Diagram

```
>>> circuit.diagram(force_separation = True)
```



15.6.3.2 States

```
>>> first_state.print()  
|x> = a|0> + b|1>
```

```
>>> second_state.print()  
|y> = c|0> + d|1>
```

```
>>> third_state.print()  
|z> = u|0> + v|1>
```

```
>>> output_state.print()  
|x, y, z ⊕ xy> = a*c*u|0,0,0> + a*c*v|0,0,1> + a*d*u|0,1,0> + a*d*v|0,1,1> + b*c*u|1,0,  
    |0> + b*c*v|1,0,1> + b*d*v|1,1,0> + b*d*u|1,1,1>
```

15.7 Half adder

15.7.1 Description

An *adder* is a circuit that sums the values of two quantum vector states in some basis. For instance, this could be the states $|x\rangle$ and $|y\rangle$ in a d -dimensional number basis $\{|n\rangle\}_{n=0}^{d-1}$. As described in *Example: CNOT* (page 207), this can be accomplished with just a CNOT (controlled-NOT) gate, which produces the sum state $|x \oplus y\rangle$.

We consider here perhaps the simplest form of a quantum adder: a *half adder* (see Figure 15.7). In addition to producing a *summation* output value

$$s = x \oplus y, \quad (15.10)$$

a half adder also produces a *carry* output value

$$c' = xy, \quad (15.11)$$

which represents the *overflow* of the summation. This is simply the value by which the modular sum of the inputs x and y is larger than the modulus d . The form of a half adder presented here was first described by Feynman [34] in the context of quantum computing.

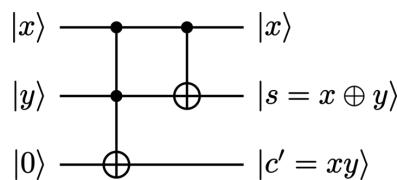


Figure 15.7: A quantum half adder.

A truth table for this circuit in the context of qubits appears in Table 15.3.

Table 15.3: Truth table for a half adder.

x (augend)	y (addend)	s (sum)	c' (carry)
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

15.7.2 Implementation

Listing 15.7: /text/examples/algorithms/adder_half.py

```

from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Not
from qhronology.quantum.circuits import QuantumCircuit

# Input
augend_state = VectorState(spec=[("a", [0]), ("b", [1])], label="x")
addend_state = VectorState(spec=[(1, [1])], label="y")
zero_state = VectorState(spec=[(1, [0])], label="0")

# Gates
CCN = Not(targets=[2], controls=[0, 1], num_systems=3)
CNI = Not(targets=[1], controls=[0], num_systems=3)
  
```

(continues on next page)

(continued from previous page)

```
# Circuit
adder = QuantumCircuit(
    inputs=[augend_state, addend_state, zero_state], gates=[CCN, CNI]
)
adder.diagram()

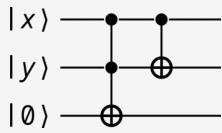
# Output
sum_state = adder.state(label="s", traces=[0, 2])
carry_output_state = adder.state(label="c'", traces=[0, 1])

# Results
augend_state.print()
addend_state.print()
sum_state.print()
carry_output_state.print()
```

15.7.3 Output

15.7.3.1 Diagram

```
>>> adder.diagram()
```



15.7.3.2 States

```
>>> augend_state.print()
|x> = a|0> + b|1>
```

```
>>> addend_state.print()
|y> = |1>
```

```
>>> sum_state.print()
s = b*conjugate(b)|0><0| + a*conjugate(a)|1><1|
```

```
>>> carry_output_state.print()
c' = a*conjugate(a)|0><0| + b*conjugate(b)|1><1|
```

15.8 Full adder

15.8.1 Description

A *full* adder is simply an adder with *three* (non-zero) inputs, as opposed to the two of a half adder. With the first and second of these inputs being the ordinary pair of values to be summed (e.g., x and y), a full adder's third input is used to modify the addition operation. This enables the circuit to take into account the carry output value c of a previous iteration, yielding the summation output

$$s = x \oplus y \oplus c, \quad (15.12)$$

and the carry output

$$c' = xy \oplus yc \oplus xc. \quad (15.13)$$

Importantly, this means that full adders can be used in succession as a way to sum multi-qubit-encoded numbers. Like the half adder, the full adder (as it is presented here in [Figure 15.8](#)) was first studied by Feynman [34].

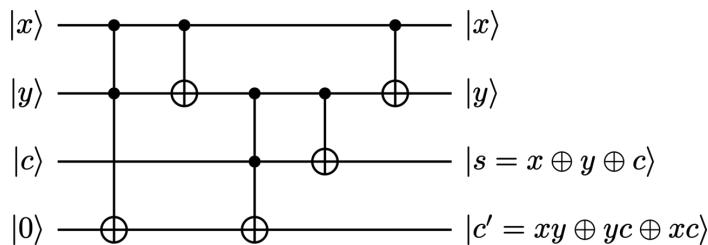


Figure 15.8: A quantum full adder.

A truth table for this circuit in the context of qubits appears in [Table 15.4](#).

Table 15.4: Truth table for the full adder.

x (augend)	y (addend)	c (carry input)	s (sum)	c' (carry output)
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

15.8.2 Implementation

Listing 15.8: /text/examples/algorithms/adder_full.py

```
from qchronology.quantum.states import VectorState
from qchronology.quantum.gates import Not
from qchronology.quantum.circuits import QuantumCircuit

augend_integers = [0, 1]
addend_integers = 1

# Input
augend_state = VectorState(
```

(continues on next page)

(continued from previous page)

```

spec=[("a", [0]), ("b", [1])],
conditions=[("a*conjugate(a) + b*conjugate(b)", 1)],
label="x",
)
addend_state = VectorState(spec=[(1, [1])], label="y")
carry_input_state = VectorState(spec=[(1, [1])], label="c")
zero_state = VectorState(spec=[(1, [0])], label="0")

# Gates
CCIN = Not(targets=[3], controls=[0, 1], num_systems=4)
CNII = Not(targets=[1], controls=[0], num_systems=4)
ICCN = Not(targets=[3], controls=[1, 2], num_systems=4)
ICNI = Not(targets=[2], controls=[1], num_systems=4)

# Circuit
adder = QuantumCircuit(
    inputs=[augend_state, addend_state, carry_input_state, zero_state],
    gates=[CCIN, CNII, ICCN, ICNI, CNII],
)
adder.diagram()

# Output
sum_state = adder.state(label="s", traces=[0, 1, 3])
carry_output_state = adder.state(label="c'", traces=[0, 1, 2])

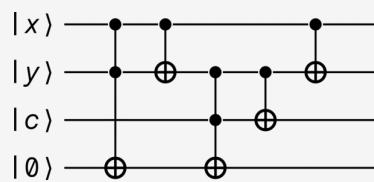
# Results
augend_state.print()
addend_state.print()
carry_input_state.print()
sum_state.print()
carry_output_state.print()

```

15.8.3 Output

15.8.3.1 Diagram

```
>>> adder.diagram()
```



15.8.3.2 States

```
>>> augend_state.print()
|x> = a|0> + b|1>
```

```
>>> addend_state.print()
|y> = |1>
```

```
>>> carry_input_state.print()
|c> = |1>
```

```
>>> sum_state.print()
s = a*conjugate(a)|0><0| + b*conjugate(b)|1><1|
```

```
>>> carry_output_state.print()
c' = |1><1|
```

15.9 Ripple-carry adder

15.9.1 Description

In this example, we use the single quantum adder circuit described in *Example: Full adder* (page 217) to construct a quantum *ripple-carry* adder. Given two (non-negative) integers x and y (encoded using qubits) as input, a ripple-carry adder computes the (encoded) arithmetic sum s of these integers, that is, $s = x + y$. In this form, the two integers are known as *summands*, and are also specifically called *augend* (x) and *addend* (y) according to the order in which they appear in the addition operation.

In the case of a single adder, the magnitudes of the summands and their sum are confined to being within the dimensionality (e.g., binary) of their single-unit (e.g., bit) encoding. Integers larger than this limit therefore require multiple units of information to be encoded. For example, a non-negative integer z can be encoded with a number N of d -dimensional unit values provided $z \leq d^N - 1$. Under this encoding, the integer can be represented using an ordered array (or set or string) of such values,

$$z \equiv (z_{N-1}, z_{N-2}, \dots, z_1, z_0) = (z_n)_{n=0}^{N-1} \quad (15.14)$$

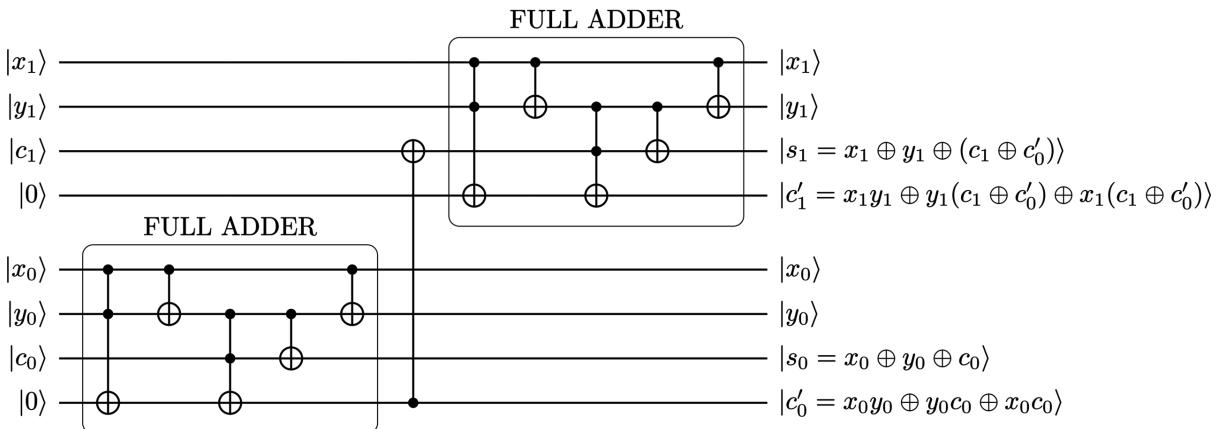
which collectively reconstruct the (decimal) value of the integer with the unique decoding expansion

$$\begin{aligned} z &= \sum_{n=0}^{N-1} z_n d^n \\ &= z_0 d^0 + z_1 d^1 + \dots + z_{N-2} d^{N-2} + z_{N-1} d^{N-1}. \end{aligned} \quad (15.15)$$

This is a type of *unsigned* integer encoding: all units of information are used to store the magnitude of the number, and there is no unit (e.g., a *sign bit*) reserved for storing the sign of the integer. It is perhaps the simplest integer encoding scheme possible, and so can be readily implemented in the context of quantum computing by utilizing the ability of quantum states to store information (both classical and quantum). This is typically realized using 2-dimensional (binary) qubits (due to their simplicity), and so a collection of N such states allows for the encoding of (non-negative) integers up to $2^N - 1$ via the scheme described in (15.14) and (15.15). The addition of any two such encoded integers, for example,

$$\begin{aligned} |x\rangle &\equiv \bigotimes_{n=N-1}^0 |x_n\rangle = |x_{N-1}, x_{N-2}, \dots, x_1, x_0\rangle, \\ |y\rangle &\equiv \bigotimes_{n=N-1}^0 |y_n\rangle = |y_{N-1}, y_{N-2}, \dots, y_1, y_0\rangle, \end{aligned} \quad (15.16)$$

can then be accomplished by a succession of exactly N quantum ripple-carry adders. Each of these adders is applied to a different pair of subsystems in the encoded integers' states—the first acts on the least significant qubit ($n = 0$), with the carry qubit from the previous adder being used as input to the next one. An example of this for a 2-qubit encoding ($N = 2$) is depicted in [Figure 15.9](#).



[Figure 15.9](#): A 2-qubit quantum ripple-carry adder. The CNOT operation between the FULL ADDER subcircuits transfers the value of the previous iteration's carry output to the carry input of the next iteration.

From this circuit, the general output values of the summation qubits can be computed to be

$$s_i = \begin{cases} x_0 \oplus y_0 \oplus c_0 & i = 0; \\ x_i \oplus y_i \oplus c_i \oplus c'_{i-1} & i > 0; \end{cases} \quad (15.17)$$

while the carry values are

$$c'_i = \begin{cases} x_0 y_0 \oplus y_0 c_0 \oplus x_0 c_0 & i = 0; \\ x_i y_i \oplus (x_i \oplus y_i)(c_i \oplus c'_{i-1}) & i > 0. \end{cases} \quad (15.18)$$

With these, the corresponding states

$$\begin{aligned} |s\rangle &\equiv \bigotimes_{n=N-1}^0 |s_n\rangle, \\ |c'\rangle &\equiv \bigotimes_{n=N-1}^0 |c'_n\rangle, \end{aligned} \quad (15.19)$$

can be decoded, using (15.15), to be

$$\begin{aligned} s &= \sum_{n=0}^{N-1} s_n d^n, \\ c' &= \sum_{n=0}^{N-1} c'_n d^n. \end{aligned} \quad (15.20)$$

15.9.2 Implementation

Due to performance constraints, this example sums two 2-bit integers. Increasing the `encoding_depth` greatly increases the calculation time as a result of having to perform operations on larger matrices. The decimal `augend_integer` and `addend_integer` variables can be freely changed, provided their summation is within the encodable range.

Listing 15.9: /text/examples/algorithms/adder_ripple.py

```
from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Not
from qhronology.quantum.circuits import QuantumCircuit
from qhronology.utilities.helpers import flatten_list
from qhronology.mechanics.matrices import encode, decode, decode_fast

import sympy as sp

import time

initial_time = time.time()

augend_integer = 1
addend_integer = 1
encoding_depth = 2

# Input
augend_state = VectorState(
    spec=encode(integer=augend_integer, num_systems=encoding_depth), label="x"
)
addend_state = VectorState(
    spec=encode(integer=addend_integer, num_systems=encoding_depth), label="y"
)
```

(continues on next page)

(continued from previous page)

```

)
augend_states = []
addend_states = []
carry_states = []
zero_states = []
for i in range(0, encoding_depth):
    target_bit = i
    complement_bits = [n for n in range(0, encoding_depth) if n != target_bit]

    augend_state.partial_trace(complement_bits)
    addend_state.partial_trace(complement_bits)

    augend_bit = decode(augend_state.output())
    addend_bit = decode(addend_state.output())

    augend_bit_state = VectorState(
        spec=encode(augend_bit, 1),
        label=augend_state.label + str(encoding_depth - 1 - i),
    )
    addend_bit_state = VectorState(
        spec=encode(addend_bit, 1),
        label=addend_state.label + str(encoding_depth - 1 - i),
    )
    carry_state = VectorState(
        spec=encode(0, 1), label="c" + str(encoding_depth - 1 - i)
    )
    zero_state = VectorState(spec=encode(0, 1), label="0")

    augend_states.append(augend_bit_state)
    addend_states.append(addend_bit_state)
    carry_states.append(carry_state)
    zero_states.append(zero_state)

    augend_state.reset()
    addend_state.reset()

input_spec = flatten_list(
    [
        [augend_states[i], addend_states[i], carry_states[i], zero_states[i]]
        for i in range(0, encoding_depth)
    ]
)

# Gates
gates = []
for i in range(0, encoding_depth):
    system_shift = 4 * (encoding_depth - 1 - i)

    CCIN = Not(
        targets=[system_shift + 3],
        controls=[system_shift + 0, system_shift + 1],
        num_systems=4 * encoding_depth,
    )
    CNII = Not(
        targets=[system_shift + 1],

```

(continues on next page)

(continued from previous page)

```

        controls=[system_shift + 0],
        num_systems=4 * encoding_depth,
    )
    ICCN = Not(
        targets=[system_shift + 3],
        controls=[system_shift + 1, system_shift + 2],
        num_systems=4 * encoding_depth,
    )
    ICNI = Not(
        targets=[system_shift + 2],
        controls=[system_shift + 1],
        num_systems=4 * encoding_depth,
    )

    adder_single = [CCIN, CNII, ICCN, ICNI, CNII]
    gates.append(adder_single)

    if i != max(range(0, encoding_depth)):
        CNOT = Not(
            targets=[system_shift - 2],
            controls=[system_shift + 3],
            num_systems=4 * encoding_depth,
        )
        adder_single.append(CNOT)

gates = flatten_list(gates)

# Circuit
adder = QuantumCircuit(inputs=input_spec, gates=gates)
adder.diagram()

# Output
sum_registers = [2 + i * 4 for i in range(0, encoding_depth)]
not_sum_registers = [i for i in range(0, 4 * encoding_depth) if i not in sum_registers]
sum_state = adder.state(label="s", traces=not_sum_registers)
sum_integer = decode_fast(sum_state.output())
sum_state = VectorState(spec=encode(sum_integer), label="s")

# Results
augend_state.print()
addend_state.print()
sum_state.print()

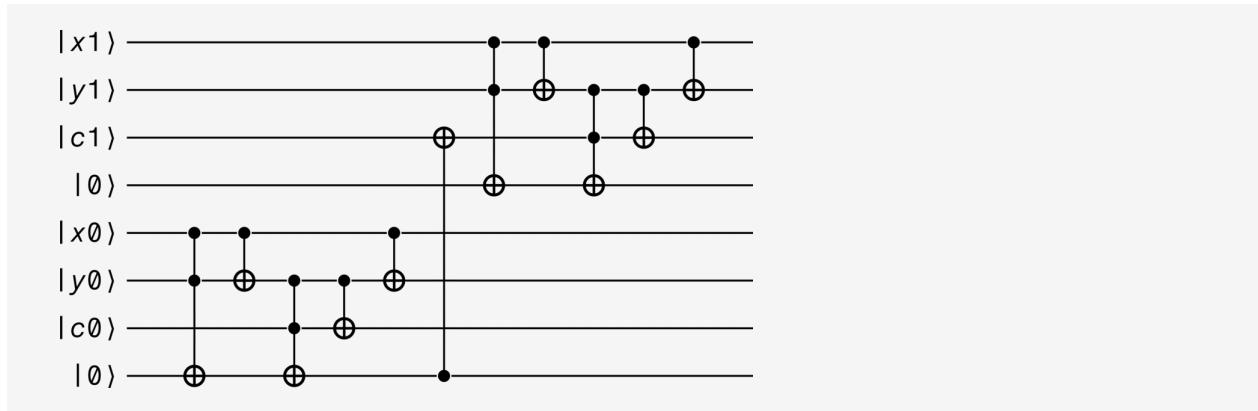
final_time = time.time()
computation = f"Computation: {augend_integer} + {addend_integer} = {sum_integer}"
duration = f"Duration: {sp.N(final_time - initial_time,8).round(3)} seconds"
print(computation)
print(duration)

```

15.9.3 Output

15.9.3.1 Diagram

```
>>> adder.diagram()
```



15.9.3.2 States

```
>>> augend_state.print()
|x> = |0,1>
```

```
>>> addend_state.print()
|y> = |0,1>
```

```
>>> sum_state.print()
|s> = |1,0>
```

15.9.3.3 Results

```
>>> print(computation)
Computation: 1 + 1 = 2
```

```
>>> print(duration)
Duration: 16.363 seconds
```

This is of course *extremely* slow, mainly due to the computations involving relatively large matrices in the underlying calculation. Optimization of Qhronology's linear algebra algorithms, particularly the partial trace implementation, should improve both efficiency and speed.

15.10 Ripple-carry adder (iterative)

15.10.1 Description

This example builds upon the four-qubit ripple-carry adder in *Example: Ripple-carry adder* (page 220). The implementation here simplifies the algorithm by instead applying copies of the circuit successively to a tetrapartite input state, which is a composition of the augend, addend, carry, and zero states. As a result, the sequence of trace operations (necessary to isolate the output sum and carry states) mixes the pure (vector) input composition, thereby destroying linearity of the evolution in the input states. This means that only mixed states for each qubit in the output sum state can be recovered, and so we can only sum single integers, not *superpositions* of integers. However, keeping the Hilbert space to a smaller total dimensionality at any one time makes the computations *much* faster, and so we can work with significantly larger integers.

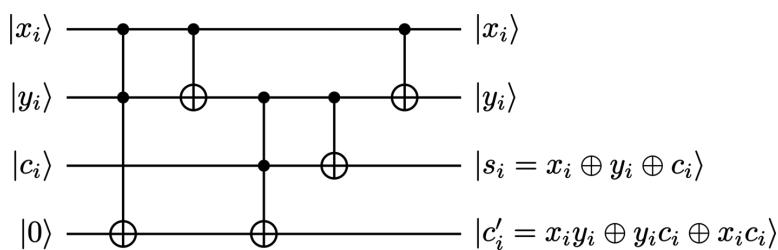


Figure 15.10: One iteration of a multi-qubit quantum ripple-carry adder.

15.10.2 Implementation

In this example, each qubit in the uppermost set on the diagram corresponds to the leftmost (most-significant) qubit in their respective encoded state, while each qubit in the lowermost set similarly corresponds to the rightmost (least-significant) qubit.

Listing 15.10: /text/examples/algorithms/adder_ripple_iterative.py

```
from qhronology.quantum.states import QuantumState, VectorState
from qhronology.quantum.gates import Not
from qhronology.quantum.circuits import QuantumCircuit
from qhronology.mechanics.matrices import encode, decode

import sympy as sp
from sympy.physics.quantum import TensorProduct

import time

initial_time = time.time()

augend_integer = 31
addend_integer = 217
encoding_depth = 8

# Input
augend_state = VectorState(
    spec=encode(integer=augend_integer, num_systems=encoding_depth), label="x_i"
)
addend_state = VectorState(
    spec=encode(integer=addend_integer, num_systems=encoding_depth), label="y_i"
)
carry_state = VectorState(spec=encode(0, 1), label="c_i")
zero_state = VectorState(spec=encode(0, 1), notation="0")
```

(continues on next page)

(continued from previous page)

```

# Gates
CCIN = Not(targets=[3], controls=[0, 1], num_systems=4)
CNII = Not(targets=[1], controls=[0], num_systems=4)
ICCN = Not(targets=[3], controls=[1, 2], num_systems=4)
ICNI = Not(targets=[2], controls=[1], num_systems=4)

# Circuits
sum_qubits = []
for i in range(encoding_depth - 1, -1, -1):
    target_bit = i
    complement_bits = [n for n in range(0, encoding_depth) if n != target_bit]

    augend_state.reset()
    addend_state.reset()

    augend_state.partial_trace(complement_bits)
    addend_state.partial_trace(complement_bits)

    adder = QuantumCircuit(
        inputs=[augend_state, addend_state, carry_state, zero_state],
        gates=[CCIN, CNII, ICCN, ICNI, CNII],
    )

    sum_qubit = adder.state(label="s", traces=[0, 1, 3])
    carry_state = adder.state(label="c_i", traces=[0, 1, 2])
    sum_qubits = [sum_qubit.output()] + sum_qubits

adder.diagram()

# Output
sum_state = QuantumState(spec=sp.Matrix(TensorProduct(*sum_qubits)), label="s")
sum_integer = decode(sum_state.output())
sum_state = VectorState(spec=encode(sum_integer), label="s")

augend_state.reset()
addend_state.reset()

augend_state.label = "x"
addend_state.label = "y"

# Results
augend_state.print()
addend_state.print()
sum_state.print()

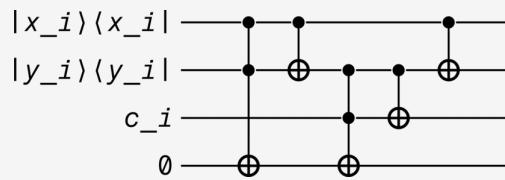
final_time = time.time()
computation = f"Computation: {augend_integer} + {addend_integer} = {sum_integer}"
duration = f"Duration: {sp.N(final_time - initial_time,8).round(3)} seconds"
print(computation)
print(duration)

```

15.10.3 Output

15.10.3.1 Diagram

```
>>> adder.diagram()
```



15.10.3.2 States

```
>>> augend_state.print()
|x> = |0,0,0,1,1,1,1,1>
```

```
>>> addend_state.print()
|y> = |1,1,0,1,1,0,0,1>
```

```
>>> sum_state.print()
|s> = |1,1,1,1,1,0,0,0>
```

15.10.3.3 Results

```
>>> print(computation)
Computation: 31 + 217 = 248
```

```
>>> print(duration)
Duration: 3.621 seconds
```

Much faster and for much larger numbers than the linear implementation in *Ripple-carry adder* (page 220).

15.11 Carry-lookahead adder

15.11.1 Description

A quantum *carry-lookahead* adder, proposed by Vedral et al. [237], is another style of multi-qubit adder and appears in Figure 15.11. It consists of two phases: in the first, the carry qubits are computed, while in the second, the sums (taking into account the value of the carry qubits) are computed (in addition to the carry qubits being reverted back to their original values). Note that in this implementation, the order of the qubits in the encoding is reversed, such that the least-significant qubits are at the top while the most-significant qubits are at the bottom.

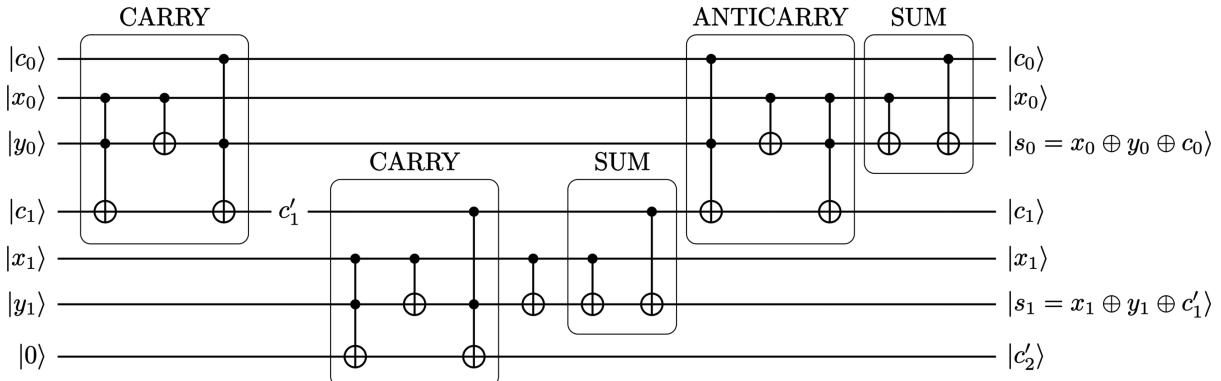


Figure 15.11: A 2-qubit quantum carry-lookahead adder.

The values of the summation qubits can be determined to be

$$s_i = \begin{cases} x_0 \oplus y_0 \oplus c_0 & i = 0; \\ x_i \oplus y_i \oplus c'_i & i > 0. \end{cases} \quad (15.21)$$

This version of an adder resets all output carry qubits to their original input values. The intermediary carry values however are

$$c'_i = \begin{cases} c_1 \oplus x_0 y_0 \oplus c_0 (x_0 \oplus y_0) & i = 1; \\ c_i \oplus x_{i-1} y_{i-1} \oplus c'_{i-1} (x_{i-1} \oplus y_{i-1}) & i > 1. \end{cases} \quad (15.22)$$

The last qubit is simply an overflow qubit and has the value

$$c'_d = c_d \oplus x_{d-1} y_{d-1} \oplus c'_{d-1} (x_{d-1} \oplus y_{d-1}). \quad (15.23)$$

It is non-zero when the sum of the most-significant qubits (plus carry) “wraps around” (due to the modular arithmetic), and so represents the case where the total resulting sum is too big to be encoded with just d qubits, a situation known as *integer overflow*. If the value of this qubit is not required, then the circuit can be simplified by removing the last CARRY, the lone CNOT, and the qubit itself.

15.11.2 Implementation

In this implementation, the dimensionality of the input state can be decreased by removing the overflow qubit (via `overflow_qubit = False`), thereby reducing the execution time of the algorithm.

Listing 15.11: /text/examples/algorithms/adder_lookahead.py

```
from qhrontology.quantum.states import VectorState
from qhrontology.quantum.gates import Not
from qhrontology.quantum.circuits import QuantumCircuit
from qhrontology.utilities.helpers import flatten_list
from qhrontology.mechanics.matrices import encode, decode
```

(continues on next page)

(continued from previous page)

```

import sympy as sp

import time

initial_time = time.time()

augend_integer = 1
addend_integer = 1
encoding_depth = 2
overflow_qubit = True

# Input
augend_state = VectorState(
    spec=encode(integer=augend_integer, num_systems=encoding_depth, reverse=True),
    label="x",
)
addend_state = VectorState(
    spec=encode(integer=addend_integer, num_systems=encoding_depth, reverse=True),
    label="y",
)
zero_state = []
if overflow_qubit is True:
    zero_state = VectorState(spec=encode(integer=0, num_systems=1), label="0")

augend_states = []
addend_states = []
carry_states = []
for i in range(0, encoding_depth):
    target_bit = i
    complement_bits = [n for n in range(0, encoding_depth) if n != target_bit]

    augend_state.partial_trace(complement_bits)
    addend_state.partial_trace(complement_bits)

    augend_bit = decode(augend_state.output())
    addend_bit = decode(addend_state.output())

    augend_bit_state = VectorState(
        spec=encode(integer=augend_bit, num_systems=1),
        label=augend_state.label + str(i),
    )
    addend_bit_state = VectorState(
        spec=encode(integer=addend_bit, num_systems=1),
        label=addend_state.label + str(i),
    )
    carry_state = VectorState(spec=encode(integer=0, num_systems=1), label="c" + str(i))

    augend_states.append(augend_bit_state)
    addend_states.append(addend_bit_state)
    carry_states.append(carry_state)

    augend_state.reset()
    addend_state.reset()

input_spec = flatten_list(

```

(continues on next page)

(continued from previous page)

```

[  

    [carry_states[i], augend_states[i], addend_states[i]]  

    for i in range(0, encoding_depth)  

]  

+ [zero_state]  

)  
  

# Gates  
  

# Construct sequence of CARRY gates  

carries = []  

for i in range(0, encoding_depth - 1 + int(overflow_qubit)):  

    system_shift = 3 * i  
  

    ICCN = Not(  

        targets=[system_shift + 3],  

        controls=[system_shift + 1, system_shift + 2],  

        num_systems=3 * encoding_depth + int(overflow_qubit),  

    )  

    ICNI = Not(  

        targets=[system_shift + 2],  

        controls=[system_shift + 1],  

        num_systems=3 * encoding_depth + int(overflow_qubit),  

    )  

    CICN = Not(  

        targets=[system_shift + 3],  

        controls=[system_shift + 0, system_shift + 2],  

        num_systems=3 * encoding_depth + int(overflow_qubit),  

    )  
  

    carry = [ICCN, ICNI, CICN]  

    carries.append(carry)  
  

CNOT = []  

if overflow_qubit is True:  

    system_shift = 3 * (encoding_depth - 1)  

    CNOT = Not(  

        targets=[system_shift + 2],  

        controls=[system_shift + 1],  

        num_systems=3 * encoding_depth + int(overflow_qubit),  

    )  
  

# Construct sequence of ANTICARRY and SUM gates  

anticarries_sums = []  

for i in range(0, encoding_depth):  

    system_shift = 3 * (encoding_depth - 1 - i)  
  

    if i != min(range(0, encoding_depth)):  

        CICN = Not(  

            targets=[system_shift + 3],  

            controls=[system_shift + 0, system_shift + 2],  

            num_systems=3 * encoding_depth + int(overflow_qubit),  

        )  

        ICNI = Not(  

            targets=[system_shift + 2],  

            controls=[system_shift + 1],  

)

```

(continues on next page)

(continued from previous page)

```

        num_systems=3 * encoding_depth + int(overflow_qubit),
    )
    ICCN = Not(
        targets=[system_shift + 3],
        controls=[system_shift + 1, system_shift + 2],
        num_systems=3 * encoding_depth + int(overflow_qubit),
    )

    anticarry = [CICN, ICNI, ICCN]
    anticarries_sums.append(anticarry)

    ICNI = Not(
        targets=[system_shift + 2],
        controls=[system_shift + 1],
        num_systems=3 * encoding_depth + int(overflow_qubit),
    )
    CINI = Not(
        targets=[system_shift + 2],
        controls=[system_shift + 0],
        num_systems=3 * encoding_depth + int(overflow_qubit),
    )

    summation = [ICNI, CINI]
    anticarries_sums.append(summation)

gates = flatten_list(carries + [CNOT] + anticarries_sums)

# Circuit
adder = QuantumCircuit(inputs=input_spec, gates=gates)
adder.diagram()

# Output
sum_registers = [3 * i + 2 for i in range(0, encoding_depth)]
sum_registers_complement = [
    i
    for i in range(0, 3 * encoding_depth + int(overflow_qubit))
    if i not in sum_registers
]
sum_state = adder.state(label="s", traces=sum_registers_complement)
sum_integer = decode(matrix=sum_state.output(), reverse=True)
sum_state = VectorState(spec=encode(integer=sum_integer, reverse=True), label="s")

# Results
augend_state.print()
addend_state.print()
sum_state.print()

final_time = time.time()
computation = f"Computation: {augend_integer} + {addend_integer} = {sum_integer}"
duration = f"Duration: {sp.N(final_time - initial_time,8).round(3)} seconds"
print(computation)
print(duration)

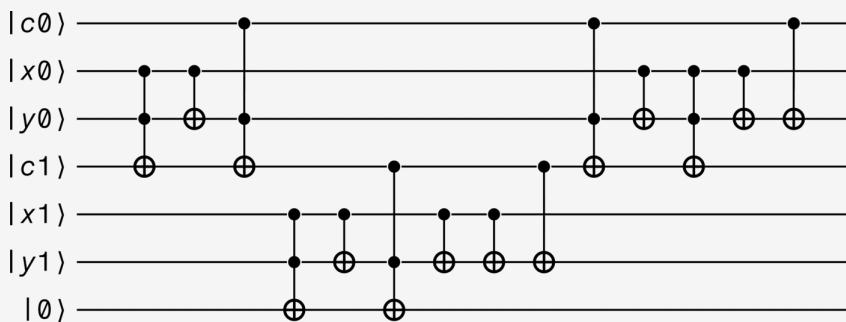
```

15.11.3 Output

15.11.3.1 Diagram

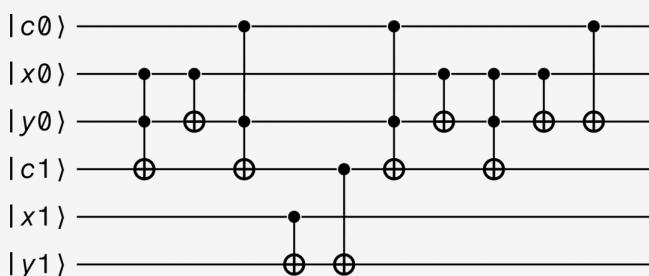
When `overflow_qubit = True`:

```
>>> adder.diagram()
```



When `overflow_qubit = False`:

```
>>> adder.diagram()
```



15.11.3.2 States

```
>>> augend_state.print()
|x> = |1,0>
```

```
>>> addend_state.print()
|y> = |1,0>
```

```
>>> sum_state.print()
|s> = |0,1>
```

15.11.3.3 Results

```
>>> print(computation)
Computation: 1 + 1 = 2
```

When `overflow_qubit = True`:

```
>>> print(duration)
Duration: 5.815 seconds
```

When `overflow_qubit = False`:

```
>>> print(duration)
Duration: 1.469 seconds
```

This version of a multi-qubit full adder is evidently much faster than the *Ripple-carry adder* (page 220) for the equivalent number of qubits, which highlights the efficiency advantage of using fewer qubits to achieve the same computation.

15.12 Fourier transform adder

15.12.1 Description

The version of a quantum adder presented here sums two multi-qubit integers essentially by computing the summation in Fourier space [238, 239, 240, 241, 242, 243]. This is achieved by first performing a quantum (discrete) Fourier transform on one of the encoded integers, following with a controlled-phase gate (serving the same function as an ordinary full adder in ordinary non-Fourier Hilbert space), and concluding with an inverse Fourier transform on the same integer.

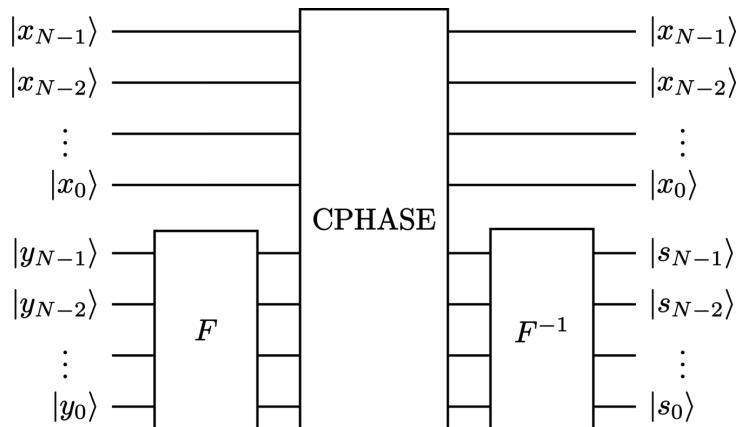


Figure 15.12: A quantum circuit diagram of a QFT-based adder.

An example of this algorithm for 3-qubit integers appears in [Figure 15.13](#).

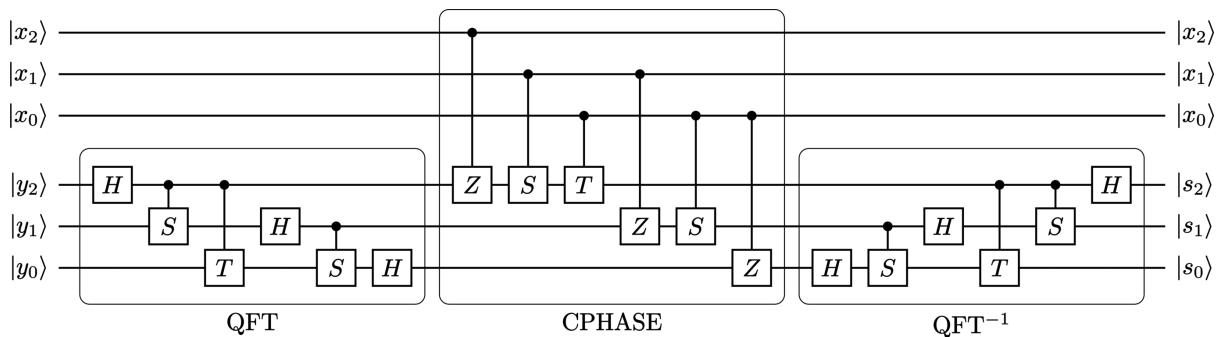


Figure 15.13: A QFT-based adder for 3-qubit states.

15.12.2 Implementation

Listing 15.12: /text/examples/algorithms/adder_fourier.py

```
from qhrontology.quantum.states import VectorState
from qhrontology.quantum.gates import Hadamard, Phase, Fourier
from qhrontology.quantum.circuits import QuantumCircuit
from qhrontology.utilities.helpers import flatten_list
from qhrontology.mechanics.matrices import encode, decode

import sympy as sp

import time

initial_time = time.time()
```

(continues on next page)

(continued from previous page)

```

augend_integer = 1
addend_integer = 1
encoding_depth = 2

# Input
augend_state = VectorState(
    spec=encode(integer=augend_integer, num_systems=encoding_depth), label="x"
)
addend_state = VectorState(
    spec=encode(integer=addend_integer, num_systems=encoding_depth), label="y"
)

QFT = []
IQFT = []
for i in range(0, encoding_depth):
    count = encoding_depth - i
    for j in range(0, count):
        if j == 0:
            QFT.append(
                Hadamard(
                    targets=[i + encoding_depth],
                    num_systems=2 * encoding_depth,
                    conjugate=False,
                    label="H",
                )
            )
            IQFT.append(
                Hadamard(
                    targets=[i + encoding_depth],
                    num_systems=2 * encoding_depth,
                    conjugate=True,
                    label="H†",
                )
            )
        else:
            QFT.append(
                Phase(
                    targets=[i + j + encoding_depth],
                    controls=[i + encoding_depth],
                    exponent=sp.Rational(1, (2**j)),
                    num_systems=2 * encoding_depth,
                    conjugate=False,
                    label=f"{{2**j}}",
                    family="GATE",
                )
            )
            IQFT.append(
                Phase(
                    targets=[i + j + encoding_depth],
                    controls=[i + encoding_depth],
                    exponent=sp.Rational(1, (2**j)),
                    num_systems=2 * encoding_depth,
                    conjugate=True,
                    label=f"{{2**j}}†",
                    family="GATE",
                )
            )

```

(continues on next page)

(continued from previous page)

```

        )
    )

IQFT = IQFT[::-1]

CPHASE = []
for i in range(0, encoding_depth):
    count = encoding_depth - i
    for j in range(0, count):
        CPHASE.append(
            Phase(
                targets=[i + encoding_depth],
                controls=[j + i],
                exponent=sp.Rational(1, (2**j)),
                num_systems=2 * encoding_depth,
                label=f"{2**j}",
                family="GATE",
            )
        )

gates = flatten_list(QFT + CPHASE + IQFT)

# Circuit
adder = QuantumCircuit(inputs=[augend_state, addend_state], gates=gates)
adder.diagram()

# Output
sum_registers = [(i + encoding_depth) for i in range(0, encoding_depth)]
sum_registers_complement = [
    i for i in range(0, 2 * encoding_depth) if i not in sum_registers
]
print(sum_registers)
sum_state = adder.state(label="s", traces=sum_registers_complement)
sum_integer = decode(sum_state.output())

# Results
augend_state.print()
addend_state.print()
sum_state.print()

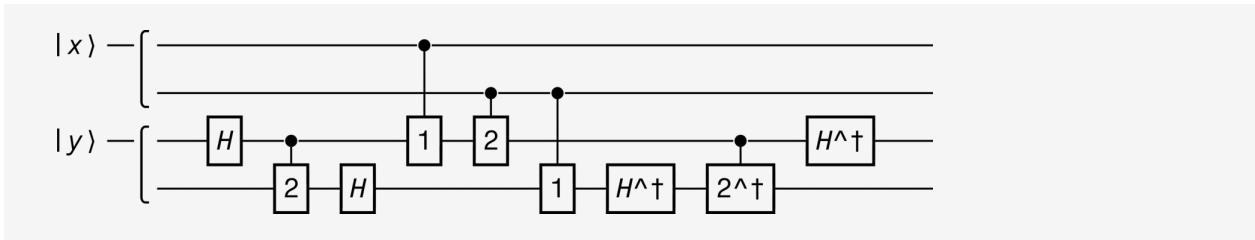
final_time = time.time()
computation = f"Computation: {augend_integer} + {addend_integer} = {sum_integer}"
duration = f"Duration: {sp.N(final_time - initial_time,8).round(3)} seconds"
print(computation)
print(duration)

```

15.12.3 Output

15.12.3.1 Diagram

```
>>> adder.diagram()
```



15.12.3.2 States

```
>>> augend_state.print()
|x⟩ = |0,1⟩
```

```
>>> addend_state.print()
|y⟩ = |0,1⟩
```

```
>>> sum_state.print()
s = |1,0⟩⟨1,0|
```

15.12.3.3 Results

```
>>> print(computation)
Computation: 1 + 1 = 2
```

```
>>> print(duration)
Duration: 0.326 seconds
```

As both the number of qubits and the circuit depth (number of gates) of the quantum Fourier adder are significantly smaller than any of the other quantum adder algorithms, we appropriately find that its operation is computationally much faster.

15.13 Quantum teleportation

15.13.1 Description

Quantum teleportation [244] is a significant technique of quantum theory in which the transfer of quantum information between two parties (that may be spatially separated) is achieved. Importantly, this does not involve the movement of physical entities, but concerns rather the transfer of the (quantum) statistics of a physical system (manifesting as a quantum state) to another. This is facilitated by a pair of entangled particles, the statistical correlations of which provide the actual mechanism of teleportation.

Note that in the process of teleporting the state, the original is destroyed, and so the no-cloning theorem remains unviolated. Additionally, because classical information needs to be sent between the two parties, the teleportation cannot occur faster than the speed of light, meaning that the law of relativity is satisfied. This example (Figure 15.14) implements the canonical version of the algorithm.

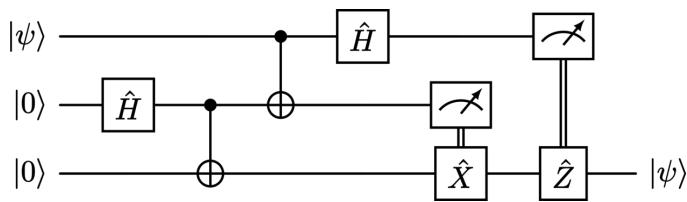


Figure 15.14: A quantum teleportation protocol.

15.13.2 Implementation

Listing 15.13: /text/examples/algorithms/teleportation.py

```
from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Hadamard, Not, Measurement, Pauli
from qhronology.quantum.circuits import QuantumCircuit
from qhronology.mechanics.matrices import ket

# Input
teleporting_state = VectorState(
    spec=[["a", "b"]],
    symbols={"a": {"complex": True}, "b": {"complex": True}},
    conditions=[("a*conjugate(a) + b*conjugate(b)", "1")],
    label=" $\psi$ ",
)
zero_state = VectorState(spec=[(1, [0, 0])], label="0,0")

# Gates
IHI = Hadamard(targets=[1], num_systems=3)
ICN = Not(targets=[2], controls=[1], num_systems=3)
CNI = Not(targets=[1], controls=[0], num_systems=3)
HII = Hadamard(targets=[0], num_systems=3)
IMI = Measurement(
    operators=[ket(0), ket(1)], observable=False, targets=[1], num_systems=3
)
MII = Measurement(
    operators=[ket(0), ket(1)], observable=False, targets=[0], num_systems=3
)
ICX = Pauli(index=1, targets=[2], controls=[1], num_systems=3)
CIZ = Pauli(index=3, targets=[2], controls=[0], num_systems=3)

# Circuit
```

(continues on next page)

(continued from previous page)

```

teleporter = QuantumCircuit(
    inputs=[teleporting_state, zero_state],
    gates=[IHI, ICN, CNI, HII, IMI, MII, ICX, CIZ],
    traces=[0, 1],
)
teleporter.diagram(force_separation=True)

# Output
teleported_state = teleporter.state(norm=1, label="ρ")

# Results
teleporting_state.print()
teleported_state.print()

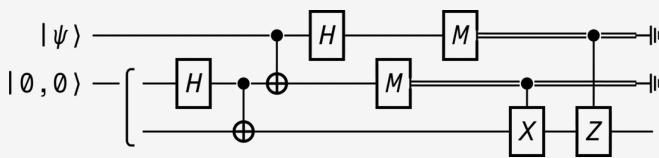
print(teleporting_state.distance(teleported_state))
print(teleporting_state.fidelity(teleported_state))

```

15.13.3 Output

15.13.3.1 Diagram

```
>>> teleporter.diagram(force_separation=True)
```



15.13.3.2 States

```
>>> teleporting_state.print()
|ψ⟩ = a|0⟩ + b|1⟩
```

```
>>> teleported_state.print()
ρ = a*conjugate(a)|0⟩⟨0| + a*conjugate(b)|0⟩⟨1| + b*conjugate(a)|1⟩⟨0| +_
+ b*conjugate(b)|1⟩⟨1|
```

15.13.3.3 Results

```
>>> teleporting_state.distance(teleported_state)
0
```

```
>>> teleporting_state.fidelity(teleported_state)
1
```

15.14 PSWAP (power-SWAP)

15.14.1 Description

Exponentiation of the SWAP gate produces an interesting interaction (PSWAP) between its input systems: the degree to which their states' values are exchanged depends entirely on the power to which the SWAP gate is taken. This effect means that the PSWAP gate is useful for modelling interesting physical phenomena such as probabilistic scattering, where the power parameter of the gate is analogous to the interaction chance of two particles in a scattering experiment. For even powers, no SWAP occurs, while for odd powers, a SWAP does occur. Non-integer values of the power therefore smoothly interpolate between these two outcomes in the form of a (weighted) quantum superposition of them. [Figure 15.15](#) depicts a simple PSWAP interaction.

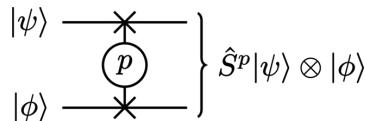


Figure 15.15: A PSWAP gate (with power p).

15.14.2 Implementation

Listing 15.14: /text/examples/algorithms/pswap.py

```
from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Swap
from qhronology.quantum.circuits import QuantumCircuit

# Input
psi = VectorState(
    spec=[("a", [0]), ("b", [1])],
    symbols={"a": {"complex": True}, "b": {"complex": True}},
    conditions=[("a*conjugate(a) + b*conjugate(b)", "1")],
    dim=2,
    norm=1,
    label="\psi",
)
phi = VectorState(
    spec=[("c", [0]), ("d", [1])],
    symbols={"c": {"complex": True}, "d": {"complex": True}},
    conditions=[("c*conjugate(c) + d*conjugate(d)", "1")],
    dim=2,
    norm=1,
    label="\phi",
)

# Gate
S = Swap(
    targets=[0, 1],
    num_systems=2,
    exponent="p",
    symbols={"p": {"real": True}},
    notation="S^p",
)

# Circuit
pswap = QuantumCircuit(inputs=[psi, phi], gates=[S])
```

(continues on next page)

(continued from previous page)

```
pswap.diagram()

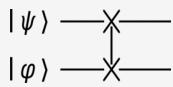
# Output
input_state = QuantumCircuit(inputs=[psi, phi]).state(label="ψ,φ")
output_state = pswap.state(label="(ψ,φ)′")

# Results
print(repr(pswap.gate()))
input_state.print()
output_state.print()
```

15.14.3 Output

15.14.3.1 Diagram

```
>>> pswap.diagram()
```



15.14.3.2 Gate

```
>>> print(repr(pswap.gate()))
Matrix([
[1, 0, 0, 0],
[0, exp(I*pi*p)/2 + 1/2, 1/2 - exp(I*pi*p)/2, 0],
[0, 1/2 - exp(I*pi*p)/2, exp(I*pi*p)/2 + 1/2, 0],
[0, 0, 0, 1]])
```

15.14.3.3 States

```
>>> input_state.print()
|ψ,φ⟩ = a*c|0,0⟩ + a*d|0,1⟩ + b*c|1,0⟩ + b*d|1,1⟩
```

```
>>> output_state.print()
|(ψ,φ)′⟩ = a*c|0,0⟩ + (a*d*(exp(I*pi*p)/2 + 1/2) + b*c*(1/2 - exp(I*pi*p)/2))|0,1⟩ +_
↪(a*d*(1/2 - exp(I*pi*p)/2) + b*c*(exp(I*pi*p)/2 + 1/2))|1,0⟩ + b*d|1,1⟩
```

15.15 iSWAP (imaginary-SWAP)

15.15.1 Description

An iSWAP (imaginary-SWAP) gate is a simply a SWAP gate in which any pair of states that are exchanged are also multiplied by a phase factor of $e^{\frac{i\pi}{2}} = i$ (the imaginary unit). It can be constructed from \hat{S} ($\hat{Z}^{\frac{1}{2}}$) gates, \hat{H} (Hadamard) gates, and CNOT gates, as depicted in [Figure 15.16](#).

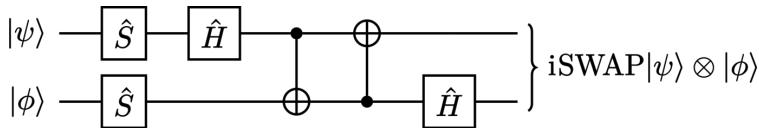


Figure 15.16: One possible construction of the iSWAP gate.

15.15.2 Implementation

Listing 15.15: /text/examples/algorithms/iswap.py

```
from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Pauli, Hadamard, Not, GateStack
from qhronology.quantum.circuits import QuantumCircuit

# Input
psi = VectorState(
    spec=[("a", [0]), ("b", [1])],
    symbols={"a": {"complex": True}, "b": {"complex": True}},
    conditions=[("a*conjugate(a) + b*conjugate(b)", "1")],
    label="\psi",
)
phi = VectorState(
    spec=[("c", [0]), ("d", [1])],
    symbols={"c": {"complex": True}, "d": {"complex": True}},
    conditions=[("c*conjugate(c) + d*conjugate(d)", "1")],
    label="\phi",
)

# Gates
S = Pauli(index=3, exponent=1 / 2, label="S")
SS = GateStack(S, S)
HI = Hadamard(targets=[0], num_systems=2)
CN = Not(targets=[1], controls=[0])
NC = Not(targets=[0], controls=[1])
IH = Hadamard(targets=[1], num_systems=2)

# Circuit
iswap = QuantumCircuit(inputs=[psi, phi], gates=[SS, HI, CN, NC, IH])
iswap.diagram()

# Output
input_state = QuantumCircuit(inputs=[psi, phi]).state(label="\psi,\phi")
output_state = iswap.state(label="(\psi,\phi)'")
output_state.simplify()

# Results
print(repr(iswap.gate(simplify=True)))
```

(continues on next page)

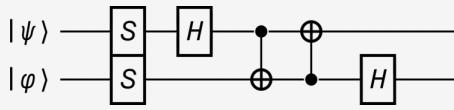
(continued from previous page)

```
input_state.print()
output_state.print()
```

15.15.3 Output

15.15.3.1 Diagram

```
>>> iswap.diagram()
```



15.15.3.2 Gate

```
>>> print(repr(iswap.gate()))
Matrix([
[1, 0, 0, 0],
[0, 0, I, 0],
[0, I, 0, 0],
[0, 0, 0, 1]])
```

15.15.3.3 States

```
>>> input_state.print()
|\psi,\phi\rangle = a*c|0,0\rangle + a*d|0,1\rangle + b*c|1,0\rangle + b*d|1,1\rangle
```

```
>>> output_state.print()
|(\psi,\phi)'\rangle = a*c|0,0\rangle + I*b*c|0,1\rangle + I*a*d|1,0\rangle + b*d|1,1\rangle
```

15.16 Quantum state tomography

15.16.1 Description

Quantum state tomography is a technique that consists of the determination of an unknown quantum state by measuring it with respect to all elements of an appropriate *tomographically* or *informationally* complete operator basis. In practise, any such basis is usually a set of observables, with some of the most prominent examples being the Pauli matrices (including the identity matrix) for qubits and the Gell-Mann matrices (also including the identity matrix) for qutrits.

Presented here is a simple tomographical reconstruction of an arbitrary qubit density matrix. Once the unknown state (or rather, an ensemble of identically prepared quantum states) has been measured exhaustively in a suitable basis, the resulting statistics (i.e., expectation values) enable the determination of the Bloch vector via (3.25), which is equivalent to precise identification of the associated (and previously unknown) state.

15.16.2 Implementation

Listing 15.16: /text/examples/algorithms/tomography_strong.py

```
from qhronology.quantum.states import MatrixState
from qhronology.quantum.gates import Pauli

import sympy as sp

import copy

# Construct symbolic density matrix
tau = sp.MatrixSymbol("τ", 2, 2).asMutable()
unknown_state = MatrixState(spec=tau, label="τ")

# Construct observables
I = Pauli(index=0, targets=[0], num_systems=1)
X = Pauli(index=1, targets=[0], num_systems=1)
Y = Pauli(index=2, targets=[0], num_systems=1)
Z = Pauli(index=3, targets=[0], num_systems=1)

# Perform measurements on the unknown state over all observables
unknown_state.measure(operators=[I, X, Y, Z], observable=True)
reconstructed_state = copy.deepcopy(unknown_state)
unknown_state.reset()
reconstructed_state.coefficient(sp.Rational(1, 2)) # Manually normalize
reconstructed_state.simplify()

# Results
unknown_state.print()
reconstructed_state.print()
```

15.16.3 Output

15.16.3.1 States

```
>>> unknown_state.print()
τ = τ[0, 0]|0⟩⟨0| + τ[0, 1]|0⟩⟨1| + τ[1, 0]|1⟩⟨0| + τ[1, 1]|1⟩⟨1|
```

```
>>> reconstructed_state.print()
τ = τ[0, 0]|0⟩⟨0| + τ[0, 1]|0⟩⟨1| + τ[1, 0]|1⟩⟨0| + τ[1, 1]|1⟩⟨1|
```

15.17 Quantum state tomography via weak measurements

15.17.1 Description

The quantum state tomography methodology presented in *Example: Quantum state tomography* (page 244) is a standard scheme that uses ordinary (“strong”) measurements—those which by necessity disturb (i.e., collapse the wave function of) the quantum system under consideration. Thus, in order to perform state tomography without (significantly) disrupting the unknown system, we can use so-called *weak measurements*, that is, quantum measurements which minimize disturbance by leaving the measured state unaffected (to first order in the strength of the measurement). The idea of performing quantum state tomography using weak measurements constitutes a powerful technique that has been used with great success in a plethora of past experimental applications [203, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272].

The scheme presented here follows the work in [214]. It is limited to *qubits*, though it is thought to be generalizable to qudits without too much difficulty. Note that while most treatments consider weak measurements characterized by weakly coupled interactions (that is, unitary operators close to identity), this formalism is based on [273, 274], in which a near eigenstate of the NOT (Pauli-X) gate is perturbed *weakly* by a control state via the standard CNOT interaction. See, e.g., [207, 213, 275, 276, 277, 278] for pedagogical exposition of weak measurements and the associated weak values, and [204, 271, 275, 279] for good examples of weak-measurement tomography.

At the heart of this methodology is an ancillary qubit, which is also known as the *target* or *probe*. In the *z*-basis $\{|0\rangle, |1\rangle\}$, it may be expressed as

$$|\chi\rangle = \sqrt{\frac{1+\epsilon}{2}}|0\rangle + \sqrt{\frac{1-\epsilon}{2}}|1\rangle, \quad 0 \leq \epsilon \leq 1. \quad (15.24)$$

Its function is to transform in response to the state of an unknown system $\hat{\tau}$. This is achieved by coupling the two states together, thereby enabling direct measurement of the ancilla to yield information regarding $\hat{\tau}$. If the interaction is sufficiently weak, then this state remains undisturbed. Note that of course, a quantum state cannot be fully characterized by performing just a single measurement. The best we can accomplish for qubits, at least using weak measurements, is to infer the expectation value of the unknown system along one of the three axes of the Bloch sphere. The original state $\hat{\tau}$ may then be completely reconstructed from the combined statistics, by way of the determination of the state’s Bloch vector.

Perhaps the simplest coupling between an unknown state and the probe that will permit this tomography scheme is the CNOT gate, which we write as

$$C^0 \hat{X}^1 \equiv |0\rangle\langle 0|^0 \otimes \hat{I}^1 + |1\rangle\langle 1|^0 \otimes \hat{\sigma}_x^1. \quad (15.25)$$

As we wish to measure along each of three Bloch axes, this can be combined with basis transformation gates $\hat{V}_k^{(\dagger)}$, yielding the system-probe unitary interaction

$$\begin{aligned} \hat{T} &= (\hat{V}_k^\dagger \otimes \hat{I}) \cdot C^0 \hat{X}^1 \cdot (\hat{V}_k \otimes \hat{I}) \\ &= |0_k\rangle\langle 0_k| \otimes \hat{I} + |1_k\rangle\langle 1_k| \otimes \hat{\sigma}_x. \end{aligned} \quad (15.26)$$

This is depicted diagrammatically as a quantum circuit in [Figure 15.17](#), where we denote the evolved ancilla by \hat{w}_k . Here, the aforementioned basis gates $\hat{V}_k^{(\dagger)}$ (where $k = 1, 2, 3$) represent transformations that map the *z*-basis eigenstates $\{|0\rangle, |1\rangle\}$ into either the *x*- ($k = 1$), *y*- ($k = 2$) or *z*- ($k = 3$) bases. Specifically, we write

$$\begin{aligned} \hat{V}_k^\dagger |0\rangle &= |0_k\rangle, \\ \hat{V}_k^\dagger |1\rangle &= |1_k\rangle, \end{aligned} \quad (15.27)$$

where $\{|0_k\rangle, |1_k\rangle\}$ are eigenstates of the three bases of our qubit Hilbert space. This means that, given the Pauli matrices (3.16), the states defined by (15.27) necessarily satisfy

$$\begin{aligned} \hat{\sigma}_k |0_k\rangle &= |0_k\rangle, \\ \hat{\sigma}_k |1_k\rangle &= -|1_k\rangle. \end{aligned} \quad (15.28)$$

For this to be fulfilled, we must have

$$\begin{aligned}\hat{V}_1 &= \hat{H}, \\ \hat{V}_2 &= \hat{H}\hat{P}^\dagger, \\ \hat{V}_3 &= \hat{I},\end{aligned}\tag{15.29}$$

where

$$\hat{H} \equiv \frac{1}{\sqrt{2}}(|0\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 0| - |1\rangle\langle 1|)\tag{15.30}$$

is the Hadamard gate, and

$$\hat{P} \equiv |0\rangle\langle 0| + i|1\rangle\langle 1|\tag{15.31}$$

is a phase shift gate.

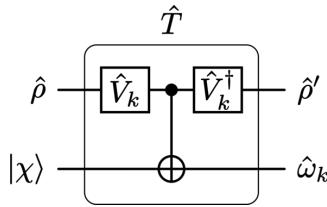


Figure 15.17: A quantum state weak-measurement tomography algorithm.

With these definitions in mind, a weak measurement can be easily computed in a number of steps. First, given the unknown system $\hat{\tau}$, the ancilla $|\chi\rangle$, and the weak-measurement unitary (15.26), the evolution of the product state $\hat{\tau} \otimes |\chi\rangle\langle\chi|$ is the standard unitary map

$$\begin{aligned}\mathbf{E}_{\hat{T}}[\hat{\tau} \otimes |\chi\rangle\langle\chi|] &= \hat{T}(\hat{\tau} \otimes |\chi\rangle\langle\chi|)\hat{T}^\dagger \\ &= \sum_{n,m=0}^1 |n_k\rangle\langle n_k| \hat{\tau}|m_k\rangle\langle m_k| \otimes \hat{\sigma}_x^n |\chi\rangle\langle\chi| \hat{\sigma}_x^m.\end{aligned}\tag{15.32}$$

The state of the evolved system $\hat{\tau}'$ after the interaction is computed by performing a partial trace on the probe's Hilbert space,

$$\begin{aligned}\hat{\tau}' &= \text{tr}_1[\hat{T}(\hat{\tau} \otimes |\chi\rangle\langle\chi|)\hat{T}^\dagger] \\ &= \sum_{n,m=0}^1 \langle\chi|\hat{\sigma}_x^m \hat{\sigma}_x^n |\chi\rangle \cdot |n_k\rangle\langle n_k| \hat{\tau}|m_k\rangle\langle m_k|.\end{aligned}\tag{15.33}$$

To show that the measurement is necessarily weak, thereby leaving this evolved state unperturbed for sufficiently small ϵ , we first use the definition of our normalized probe state (15.24) to compute

$$\langle\chi|\hat{\sigma}_x|\chi\rangle = \sqrt{1 - \epsilon^2}.\tag{15.34}$$

A Taylor series expansion of this lets us write

$$\sqrt{1 - \epsilon^2} = 1 - \frac{\epsilon^2}{2} + \mathcal{O}(\epsilon^4),\tag{15.35}$$

with which we may express the evolved system (15.33) as

$$\hat{\tau}' = \sum_{n,m=0}^1 |n_k\rangle\langle n_k| \hat{\tau}|m_k\rangle\langle m_k| + \mathcal{O}(\epsilon^2).\tag{15.36}$$

By comparing this to our probe density, which can be expanded as

$$|\chi\rangle\langle\chi| = |+\rangle\langle+| + \frac{\epsilon}{2}\hat{\sigma}_z + \mathcal{O}(\epsilon^2),\tag{15.37}$$

it is easy to see that for sufficiently small ϵ , e.g., $|\epsilon| \ll 1$ such that $\mathcal{O}(\epsilon^2) \approx 0$, the system state (15.36) approximates its unperturbed form,

$$\begin{aligned}\hat{\tau}' &\approx \sum_{n,m=0}^1 |n_k\rangle\langle n_k|\hat{\tau}|m_k\rangle\langle m_k| \\ &= \hat{\tau},\end{aligned}\tag{15.38}$$

while the probe (15.37), containing a first-order power of ϵ , becomes

$$|\chi\rangle\langle\chi| \approx |+\rangle\langle+| + \frac{\epsilon}{2}\hat{\sigma}_z.\tag{15.39}$$

This is exactly what we desire, since if the probe were to become exactly an eigenstate of the Pauli-X operator $\hat{\sigma}_x$, i.e.,

$$|\pm\rangle \equiv \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle),\tag{15.40}$$

then it would not be perturbed by the CNOT gate in its coupling (15.26) with the unknown system state. In other words, it would be unable to provide any information regarding $\hat{\tau}$, and would thus be tomographically useless. As however the probe state (15.37) does not assume the form of an eigenstate of $\hat{\sigma}_x$ while the system remains unperturbed by the interaction (provided ϵ is sufficiently small), then we can in principle infer the state of $\hat{\tau}$ via strong measurements of the evolved ancilla state $\hat{\omega}_k$ (for all k). This is the meaning of a weak measurement, and we call ϵ the *strength* of the measurement.

Let us now demonstrate how the qubit system $\hat{\tau}$ may be reconstructed by measuring the probe in the z basis. Starting with (15.32), we perform a partial trace on the first subsystem to obtain the evolved probe state,

$$\begin{aligned}\hat{\omega}_k &\equiv \text{tr}_0[\hat{T}(\hat{\tau} \otimes |\chi\rangle\langle\chi|)\hat{T}^\dagger] \\ &= \sum_{n=0}^1 \langle n_k|\hat{\tau}|n_k\rangle \cdot \hat{\sigma}_x^n |\chi\rangle\langle\chi| \hat{\sigma}_x^n.\end{aligned}\tag{15.41}$$

Computing the expectation value of $\hat{\sigma}_z$ for this form then yields

$$\begin{aligned}\text{tr}[\hat{\sigma}_z \hat{\omega}_k] &= \langle 0|\hat{\omega}_k|0\rangle - \langle 1|\hat{\omega}_k|1\rangle \\ &= \epsilon(\langle 0_k|\hat{\tau}|0_k\rangle - \langle 1_k|\hat{\tau}|1_k\rangle) \\ &= \epsilon \text{tr}[\hat{\sigma}_k \hat{\tau}],\end{aligned}\tag{15.42}$$

where we used the fact that

$$\hat{\sigma}_k = |0_k\rangle\langle 0_k| - |1_k\rangle\langle 1_k|.\tag{15.43}$$

Rescaling by a factor $\frac{1}{\epsilon}$ then allows us to make the connection between performing a z -basis measurement on the probe state (15.41) and inferring the expectation value of $\hat{\sigma}_k$ for the state of the unknown system. This is because, up to the factor ϵ , the two operations are equivalent. Note that, for brevity, we will write

$$r_k \equiv \frac{1}{\epsilon} \text{tr}[\hat{\sigma}_z \hat{\omega}_k] = \text{tr}[\hat{\sigma}_k \hat{\tau}],\tag{15.44}$$

which is often denoted in the literature as $\langle \hat{\sigma}_k \rangle$.

If we define $\hat{\sigma}_0 \equiv \hat{I}$, then the set of operators $\{\hat{\sigma}_k\}_{k=0}^3$ forms a complete basis for the space of complex 2×2 matrices. Since this coincides with the space of linear operators $\mathcal{L}(\mathcal{H})$ on our qubit (2-dimensional) Hilbert space \mathcal{H} , then any state $\hat{\tau} \in \mathcal{L}(\mathcal{H})$ can accordingly be reconstructed via the linear combination,

$$\hat{\tau} = \frac{1}{2} \sum_{k=0}^3 \text{tr}[\hat{\sigma}_k \hat{\tau}] \hat{\sigma}_k.\tag{15.45}$$

Since the coefficient for $k = 0$ is fixed by normalization (i.e., $\text{tr}[\hat{\sigma}_0 \hat{\tau}] = 1$), then the state is fully determined by the *Bloch vector*, which is defined, for an d -dimensional state, as the $(d^2 - 1)$ -dimensional

real vector of parameters $\text{tr}[\hat{\sigma}_k \tau]$ (for $k \in \mathbb{Z}_1^{d^2-1}$). Accordingly, the linear combination (15.45) is said to be the *Bloch sphere representation* of the state τ . Thus, since we have shown that we can infer expectation values (15.44) for an unknown state by coupling it with a probe via a CNOT and subsequently performing a measurement of the probe in the z -basis, then we can characterize the state completely without directly measuring it.

15.17.2 Implementation

Listing 15.17: /text/examples/algorithms/tomography_weak.py

```
from qhronology.quantum.states import VectorState, MatrixState
from qhronology.quantum.gates import Not, Hadamard, Phase, Pauli
from qhronology.quantum.circuits import QuantumCircuit
from qhronology.utilities.helpers import flatten_list

import sympy as sp

# Input
tau = sp.MatrixSymbol("τ", 2, 2).asMutable()
unknown_state = MatrixState(spec=tau, label="τ")

probe_state = VectorState(
    spec=[("sqrt((1 + ε)/2)", [0]), ("sqrt((1 - ε)/2)", [1])],
    symbols={"ε": {"real": True, "nonnegative": True}},
    conditions=[
        ("ε", "1 - ε"),
        ("ε", "1 - ε"),
    ], # Describes the fact that ε is between 0 and 1 (inclusive)
    label="χ",
)

# Gates
CN = Not(targets=[1], controls=[0], num_systems=2)
HI = Hadamard(targets=[0], num_systems=2)
PI = Phase(exponent=1 / 2, targets=[0], num_systems=2)

VI = [[HI], [PI, HI], []]

# Circuits
evolved_probe_states = []
for gates in VI:
    gates_pre = gates
    gates_post = gates[::-1]
    for gate in gates_post:
        gate.conjugate = True

    tomography = QuantumCircuit(
        inputs=[unknown_state, probe_state],
        gates=flatten_list([gates_pre, CN, gates_post]),
        traces=[0],
    )
    tomography.diagram()
    evolved_probe_state = tomography.state(label="ω")
    evolved_probe_states.append(evolved_probe_state)

# Weak-measurement quantum state tomography
Z = Pauli(index=3, targets=[0], num_systems=1)
```

(continues on next page)

(continued from previous page)

```

expectations = []
expectations.append(1) # Expectation value corresponding to the identity operator
for state in evolved_probe_states:
    state.coefficient("1/ε")
    statistics = state.measure(
        operators=[Z], targets=[0], statistics=True, observable=True
    )
    expectation = statistics[0]
    expectations.append(expectation)

# Reconstruct state via Bloch-sphere representation using weak measurements
pauli_matrices = [
    Pauli(index=i, targets=[0], num_systems=1).output() for i in range(0, 4)
]
spec = sp.zeros(2)
for i in range(0, 4):
    spec += expectations[i] * pauli_matrices[i]
reconstructed_state = MatrixState(
    spec=sp.Matrix(spec),
    conditions=[(tau[1, 1], 1 - tau[0, 0]), (1 - tau[0, 0], tau[1, 1])],
    label="τ",
)
reconstructed_state.coefficient(sp.Rational(1, 2)) # Manually normalize
reconstructed_state.simplify()

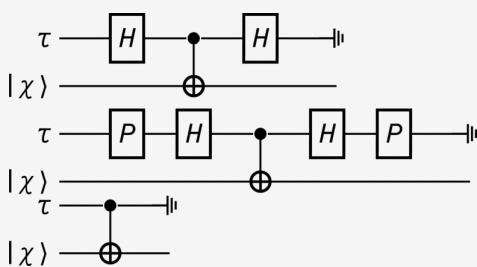
# Results
unknown_state.print()
reconstructed_state.print()

```

15.17.3 Output

15.17.3.1 Diagram

```
>>> tomography.diagram()
```



15.17.3.2 States

```
>>> unknown_state.print()
τ = τ[0, 0]|0⟩⟨0| + τ[0, 1]|0⟩⟨1| + τ[1, 0]|1⟩⟨0| + τ[1, 1]|1⟩⟨1|
```

```
>>> reconstructed_state.print()
τ = τ[0, 0]|0⟩⟨0| + τ[0, 1]|0⟩⟨1| + τ[1, 0]|1⟩⟨0| + τ[1, 1]|1⟩⟨1|
```


16.1 Grandfather paradox

16.1.1 Description

Probably the most prominent time-travel paradox is the *grandfather paradox*. Presented here is a simplified, quantum version of the paradox (based on [116]), in which a qubit CV state has an apparently paradoxical influence on its past self (exterior to the CTC) via a CNOT operation. This is illustrated in Figure 16.1.

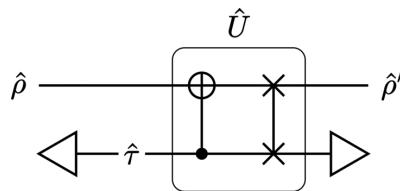


Figure 16.1: A quantum model of the grandfather paradox.

It consists of two systems: the first (upper) is the *chronology-respecting* (CR) system, while the second (lower) is the *chronology-violating* (CV) one. The Hilbert spaces of both will be ordinary qubit spaces, and using these we wish to describe the propagation of an observer (who can be either alive or dead) through the time machine and its chronology-violating region. Without loss of generality, we arbitrarily associate the level $|0\rangle$ with observer's state of being dead, while $|1\rangle$ will denote their alive state. The circuit therefore describes the propagation of an observer whose state changes in accordance with their future state in the time machine (due to the CNOT gate), after which they SWAP onto the CV system and experience the interaction from the other perspective. The combined unitary gate thus has the form

$$\hat{U} = \hat{S}^{0,1} \cdot C^1 \hat{X}^0. \quad (16.1)$$

As this is a quantum temporal paradox, there are multiple *prescriptions* (page 63) by which resolutions can be computed.

16.1.2 Solutions

16.1.2.1 D-CTCs

In Deutsch's model (D-CTCs), we necessarily have to calculate fixed-point solutions $\hat{\tau}$ for the CV system. This is accomplished by first computing the standard evolution of the bipartite input state $\hat{\rho} \otimes \hat{\tau}$,

$$\begin{aligned} \mathbf{E}_{\hat{U}}[\hat{\rho} \otimes \hat{\tau}] &= \hat{U}(\hat{\rho} \otimes \hat{\tau})\hat{U}^\dagger \\ &= \hat{S}^{0,1} \cdot C^1 \hat{X}^0 (\hat{\rho} \otimes \hat{\tau}) C^1 \hat{X}^{\dagger 0} \cdot \hat{S}^{\dagger 0,1} \\ &= \hat{S}^{0,1} (\hat{\rho} \otimes |0\rangle\langle 0| \hat{\tau} |0\rangle\langle 0| \\ &\quad + \hat{\rho}\hat{\sigma}_x^\dagger \otimes |0\rangle\langle 0| \hat{\tau}|1\rangle\langle 1| \\ &\quad + \hat{\sigma}_x\hat{\rho} \otimes |1\rangle\langle 1| \hat{\tau} |0\rangle\langle 0| \\ &\quad + \hat{\sigma}_x\hat{\rho}\hat{\sigma}_x^\dagger \otimes |1\rangle\langle 1| \hat{\tau}|1\rangle\langle 1|) \hat{S}^{\dagger 0,1} \\ &= |0\rangle\langle 0| \hat{\tau} |0\rangle\langle 0| \otimes \hat{\rho} \\ &\quad + |0\rangle\langle 0| \hat{\tau}|1\rangle\langle 1| \otimes \hat{\rho}\hat{\sigma}_x^\dagger \\ &\quad + |1\rangle\langle 1| \hat{\tau} |0\rangle\langle 0| \otimes \hat{\sigma}_x\hat{\rho} \\ &\quad + |1\rangle\langle 1| \hat{\tau}|1\rangle\langle 1| \otimes \hat{\sigma}_x\hat{\rho}\hat{\sigma}_x^\dagger. \end{aligned} \quad (16.2)$$

Subsequently tracing out the CR system yields the D-CTCs CV map,

$$\begin{aligned}\mathbf{d}_{\hat{U}}[\hat{\rho}, \hat{\tau}] &= \text{tr}_0[\hat{U}(\hat{\rho} \otimes \hat{\tau})\hat{U}^\dagger] \\ &= \langle 0|\hat{\tau}|0\rangle \cdot \hat{\rho} + \langle 1|\hat{\tau}|1\rangle \cdot \hat{\sigma}_x \hat{\rho} \hat{\sigma}_x^\dagger,\end{aligned}\quad (16.3)$$

while tracing out the CV system yields the CR map,

$$\begin{aligned}\mathbf{D}_{\hat{U}}[\hat{\rho}, \hat{\tau}] &= \text{tr}_1[\hat{U}(\hat{\rho} \otimes \hat{\tau})\hat{U}^\dagger] \\ &= |0\rangle\langle 0|\hat{\tau}|0\rangle\langle 0| \\ &\quad + |0\rangle\langle 0|\hat{\tau}|1\rangle\langle 1| \cdot [\langle 0|\hat{\rho}|1\rangle + \langle 1|\hat{\rho}|0\rangle] \\ &\quad + |1\rangle\langle 1|\hat{\tau}|0\rangle\langle 0| \cdot [\langle 1|\hat{\rho}|0\rangle + \langle 0|\hat{\rho}|1\rangle] \\ &\quad + |1\rangle\langle 1|\hat{\tau}|1\rangle\langle 1|.\end{aligned}\quad (16.4)$$

Solutions for the CV system are then given as fixed points of the corresponding map, i.e. $\hat{\tau}_{\mathbf{D}} = \mathbf{d}_{\hat{U}}[\hat{\rho}, \hat{\tau}_{\mathbf{D}}]$, and accordingly we find the unique state

$$\begin{aligned}\hat{\tau}_{\mathbf{D}} &= \frac{1}{2}\hat{\rho} + \frac{1}{2}\hat{\sigma}_x \hat{\rho} \hat{\sigma}_x^\dagger \\ &= \frac{1}{2}\left[|0\rangle\langle 0| + (\langle 0|\hat{\rho}|1\rangle + \langle 1|\hat{\rho}|0\rangle)(|0\rangle\langle 1| + |1\rangle\langle 0|) + |1\rangle\langle 1|\right].\end{aligned}\quad (16.5)$$

With this, the solution for the CR system can be computed to be

$$\hat{\rho}_{\mathbf{D}} = \frac{1}{2}\left[|0\rangle\langle 0| + (\langle 0|\hat{\rho}|1\rangle + \langle 1|\hat{\rho}|0\rangle)^2(|0\rangle\langle 1| + |1\rangle\langle 0|) + |1\rangle\langle 1|\right]\quad (16.6)$$

which is similarly unique.

16.1.2.2 P-CTCs

To compute the resolution to the grandfather paradox according to the postselected teleportation (P-CTCs) prescription of quantum time travel, we first calculate the reduced P-CTC operator,

$$\begin{aligned}\hat{\mathcal{P}} &\equiv \text{tr}_{\text{CV}}[\hat{U}] \\ &= \text{tr}_1[\hat{S}^{0,1} \cdot C^1 \hat{X}^0] \\ &= \text{tr}_1[\hat{S}^{0,1}(\hat{I} \otimes |0\rangle\langle 0| + \hat{\sigma}_x \otimes |1\rangle\langle 1|)] \\ &= \text{tr}_1[\hat{S}^{0,1}((|0\rangle\langle 0| + |1\rangle\langle 1|) \otimes |0\rangle\langle 0| + (|0\rangle\langle 1| + |1\rangle\langle 0|) \otimes |1\rangle\langle 1|)] \\ &= \text{tr}_1[|0\rangle\langle 0| \otimes |0\rangle\langle 0| + |0\rangle\langle 1| \otimes |1\rangle\langle 0| + |1\rangle\langle 1| \otimes |0\rangle\langle 1| + |1\rangle\langle 0| \otimes |1\rangle\langle 1|] \\ &= |0\rangle\langle 0| + |1\rangle\langle 0| \\ &= |+\rangle\langle +|.\end{aligned}\quad (16.7)$$

Using this, we can obtain the P-CTC CR state from its definition (7.31),

$$\begin{aligned}\hat{\rho}_{\mathbf{P}} &= \mathbf{P}_{\hat{U}}[\hat{\rho}] \\ &= \frac{\hat{\mathcal{P}}\hat{\rho}\hat{\mathcal{P}}^\dagger}{\text{tr}[\hat{\mathcal{P}}\hat{\rho}\hat{\mathcal{P}}^\dagger]} \\ &= |+\rangle\langle +|\end{aligned}\quad (16.8)$$

where we have used the canonical form

$$|+\rangle \equiv \frac{1}{\sqrt{2}}[|0\rangle + |1\rangle].\quad (16.9)$$

Finally, using the expression (7.34), we can compute the P-CTC CV state (via weak-measurement quantum state tomography) to be

$$\begin{aligned}\hat{\tau}_{\mathbf{P}} &= \frac{1}{2}\hat{\rho} + \frac{1}{2}\hat{\sigma}_x \hat{\rho} \hat{\sigma}_x^\dagger \\ &= \frac{1}{2}\left[|0\rangle\langle 0| + (\langle 0|\hat{\rho}|1\rangle + \langle 1|\hat{\rho}|0\rangle)(|0\rangle\langle 1| + |1\rangle\langle 0|) + |1\rangle\langle 1|\right],\end{aligned}\quad (16.10)$$

Note that this is identical to the D-CTC CV state (16.5).

16.1.2.3 P-CTC restrictions on initial data

The grandfather paradox is particularly interesting because it exhibits pathological behaviour under the P-CTC prescription. Indeed, its antichronological effect on a general qubit can be observed by analyzing the evolution of the entangled state,

$$|\phi\rangle_{\text{rec,CR}} = \sqrt{\Omega}|0\rangle_{\text{rec}} \otimes |0\rangle_{\text{CR}} + \sqrt{1-\Omega}|1\rangle_{\text{rec}} \otimes |1\rangle_{\text{CR}}, \quad (16.11)$$

where $0 \leq \Omega \leq 1$ controls the balance of the entanglement. Here, the first subsystem is the *record* mode which, due to the entanglement, tells us (after measurement) which CR state interacted with the P-CTC. Accordingly, by using the bipartite unitary

$$\hat{I}_{\text{rec}} \otimes \text{tr}_{\text{CV}}[\hat{U}] = \hat{I}_{\text{rec}} \otimes |+\rangle\langle 0|_{\text{CR}}, \quad (16.12)$$

where $\hat{P} = \text{tr}_{\text{CV}}[\hat{U}]$ is the P-CTC operator, we obtain the evolution

$$(\hat{I}_{\text{rec}} \otimes \text{tr}_{\text{CV}}[\hat{U}])|\phi\rangle = \sqrt{2\Omega}|0\rangle_{\text{rec}} \otimes |+\rangle_{\text{CR}}. \quad (16.13)$$

Renormalization vanishes the coefficient $\sqrt{2\Omega}$, leaving the record to consist of only the state $|0\rangle$ regardless of the value of Ω . This suggests that the CR input was likewise only $|0\rangle$ even if $\Omega \neq 1$, indicating that the entanglement was broken.

Prior to the renormalization, a value of $\Omega = 0$ leads to a vanishing P-CTC evolution, and so the renormalization process merely serves to produce a physical output state for all Ω . In doing so however, the record reveals that a CR input of $|1\rangle$ was never prepared, even when $\Omega \neq 1$. An input of $\hat{\rho} = |1\rangle\langle 1|$ is thus forbidden under the P-CTC description of this simplified grandfather paradox. With the interpretation of $|1\rangle$ as corresponding to “alive”, this result is curious, but is probably just an indication that this particular version of the grandfather paradox is overly simplistic. Additionally, in practice, the effects of quantum noise and decoherence would ensure that such an exact input state is not physically realizable. In any case, we conclude that although the grandfather paradox’s interaction places theoretical constraints for certain input CR states, a (unique) P-CTC CV state $\hat{\tau}_P$ can *always* be determined via (7.34).

16.1.3 Implementation

Listing 16.1: /text/examples/ctcs/grandfather.py

```
from qchronology.quantum.states import MixedState
from qchronology.quantum.gates import Not, Swap
from qchronology.quantum.prescriptions import QuantumCTC, DCTC, PCTC

import sympy as sp

# Input
rho = sp.MatrixSymbol("rho", 2, 2).asMutable()
input_state = MixedState(spec=rho, conditions=[(rho[0, 0], 1 - rho[1, 1])], label="rho")

# Gates
NC = Not(targets=[0], controls=[1], num_systems=2)
S = Swap(targets=[0, 1], num_systems=2)

# CTC
grandfather = QuantumCTC([input_state], gates=[NC, S], systems_respecting=[0])
grandfather.diagram()

# Output
# D-CTCs
grandfather_DCTC = DCTC(circuit=grandfather)
grandfather_DCTC_respecting = grandfather_DCTC.state_respecting(norm=False, label="rho_D")
```

(continues on next page)

(continued from previous page)

```

grandfather_DCTC_violating = grandfather_DCTC.state_violating(norm=False, label="τ_D")
grandfather_DCTC_respecting.apply(sp.factor)

# P-CTCs
grandfather_PCTC = PCTC(circuit=grandfather)
grandfather_PCTC_respecting = grandfather_PCTC.state_respecting(norm=1, label="ρ_P")
grandfather_PCTC_violating = grandfather_PCTC.state_violating(norm=1, label="τ_P")
grandfather_PCTC_respecting.coefficient("1/(sqrt(2))") # Manually renormalize
grandfather_PCTC_respecting.simplify()
grandfather_PCTC_violating.simplify()

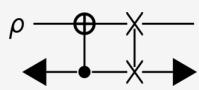
# Results
grandfather_DCTC_respecting.print()
grandfather_DCTC_violating.print()
grandfather_PCTC_respecting.print()
grandfather_PCTC_violating.print()

```

16.1.4 Output

16.1.4.1 Diagram

```
>>> grandfather.diagram()
```



16.1.4.2 States

```

>>> grandfather_DCTC_respecting.print()
ρ_D = 1/2|0⟩⟨0| + (ρ[0, 1] + ρ[1, 0])**2/2|0⟩⟨1| + (ρ[0, 1] + ρ[1, 0])**2/2|1⟩⟨0| + 1/
    ↵2|1⟩⟨1|

```

```

>>> grandfather_DCTC_violating.print()
τ_D = 1/2|0⟩⟨0| + (ρ[0, 1]/2 + ρ[1, 0]/2)|0⟩⟨1| + (ρ[0, 1]/2 + ρ[1, 0]/2)|1⟩⟨0| + 1/
    ↵2|1⟩⟨1|

```

```

>>> grandfather_PCTC_respecting.print()
ρ_P = 1/2|0⟩⟨0| + 1/2|0⟩⟨1| + 1/2|1⟩⟨0| + 1/2|1⟩⟨1|

```

```

>>> grandfather_PCTC_violating.print()
τ_P = 1/2|0⟩⟨0| + (ρ[0, 1]/2 + ρ[1, 0]/2)|0⟩⟨1| + (ρ[0, 1]/2 + ρ[1, 0]/2)|1⟩⟨0| + 1/
    ↵2|1⟩⟨1|

```

16.2 Unproven-theorem paradox

16.2.1 Description

The *unproven-theorem paradox* [23, 116, 184] (depicted in Figure 16.2) is, as we shall see, not as straightforward as other temporal paradoxes. Here, the CR system is comprised of two subsystems: the first (upper) is the *book*, and the second (lower) is the *mathematician*.

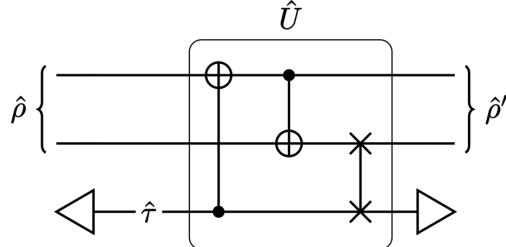


Figure 16.2: A quantum model of the unproven-theorem paradox under the D-CTCs prescription.

Described by the unitary

$$\hat{U} = \hat{S}^{1,2} \cdot C^0 \hat{X}^1 \cdot C^2 \hat{X}^0, \quad (16.14)$$

the unproven-theorem paradox encapsulates the scenario in which the proof of a theorem that a mathematician reads in the past is the very same proof which they write down in the future book (with the book having travelled back in time). We therefore denote the absence and presence of the proof in each subsystem intuitively by $|0\rangle$ and $|1\rangle$, respectively. Under this interpretation, when the mathematician and/or the book are in the state $|1\rangle$, they possess a “correct” proof, while being in the state $|0\rangle$ means that they do not. (Alternatively, one can think of these qubit levels as denoting different answers to some problem the mathematician is working on. They can then encode their proof of some theorem into an N -bit binary string by using N copies of the circuit [23].) Thus, the CR input state

$$\hat{\rho} = |0\rangle\langle 0| \otimes |0\rangle\langle 0| \quad (16.15)$$

describes the situation in which both the book and mathematician do not initially possess the proof.

16.2.2 Solutions

The D-CTC can be calculated to be

$$\begin{aligned} \hat{\rho}_D &= g|0\rangle\langle 0| \otimes |0\rangle\langle 0| + (1-g)|1\rangle\langle 1| \otimes |1\rangle\langle 1|, \\ \hat{\tau}_D &= g|0\rangle\langle 0| + (1-g)|1\rangle\langle 1|, \quad 0 \leq g \leq 1; \end{aligned} \quad (16.16)$$

while the P-CTC solutions are

$$\begin{aligned} |\psi_P\rangle &= |\Phi^+\rangle = \frac{1}{\sqrt{2}}(|0\rangle\otimes|0\rangle + |1\rangle\otimes|1\rangle), \\ \hat{\tau}_P &= \frac{1}{2}\hat{I} = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|). \end{aligned} \quad (16.17)$$

Observe that the D-CTC solution turns out to be non-unique (with free parameter g), while the P-CTC CV state is not. The interpretation of these solutions is that, according the P-CTCs, the mathematician and the book will be in a superposition of both possessing the proof and neither possessing it, while according to D-CTCs, the two parties will be in a parametrized mixture of such possibilities. Additionally, it is easy to see that the latter is the maximally entropic form of its D-CTC counterpart, which means that the ECP end-state evolution (the “correct” D-CTC fixed point which resolves the uniqueness ambiguity) in this case coincidentally is exactly the P-CTC CV solution.

16.2.3 Implementation

Listing 16.2: /text/examples/ctcs/unproven.py

```

from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import Not, Swap
from qhronology.quantum.prescriptions import QuantumCTC, DCTC, PCTC

# Input
mathematician_state = VectorState(spec=[(1, [0])], label="0")
book_state = VectorState(spec=[(1, [0])], label="0")

# Gates
NIC = Not(targets=[0], controls=[2], num_systems=3)
CNI = Not(targets=[1], controls=[0], num_systems=3)
IS = Swap(targets=[1, 2], num_systems=3)

# CTC
unproven = QuantumCTC(
    inputs=[mathematician_state, book_state],
    gates=[NIC, CNI, IS],
    systems_respecting=[0, 1],
)
unproven.diagram()

# Output
# D-CTCs
unproven_DCTC = DCTC(circuit=unproven)
unproven_DCTC_respecting = unproven_DCTC.state_respecting(norm=1, label=" $\rho_D$ ")
unproven_DCTC_violating = unproven_DCTC.state_violating(norm=1, label=" $\tau_D$ ")

# P-CTCs
unproven_PCTC = PCTC(circuit=unproven)
unproven_PCTC_respecting = unproven_PCTC.state_respecting(norm=1, label=" $\psi_P$ ")
unproven_PCTC_violating = unproven_PCTC.state_violating(norm=1, label=" $\tau_P$ ")

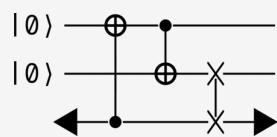
# Results
unproven_DCTC_respecting.print()
unproven_DCTC_violating.print()
unproven_PCTC_respecting.print()
unproven_PCTC_violating.print()

```

16.2.4 Output

16.2.4.1 Diagram

```
>>> unproven.diagram()
```



16.2.4.2 States

```
>>> unproven_DCTC_respecting.print()
ρ_D = g|0,0⟩⟨0,0| + (1 - g)|1,1⟩⟨1,1|
```

```
>>> unproven_DCTC_violating.print()
τ_D = g|0⟩⟨0| + (1 - g)|1⟩⟨1|
```

```
>>> unproven_PCTC_respecting.print()
|ψ_P⟩ = sqrt(2)/2|0,0⟩ + sqrt(2)/2|1,1⟩
```

```
>>> unproven_PCTC_violating.print()
τ_P = 1/2|0⟩⟨0| + 1/2|1⟩⟨1|
```

16.3 Billiard-ball paradox

16.3.1 Description

In this example, a quantum model of the billiard-ball paradox (see *Time-travelling billiard balls* (page 61)) is studied, with the analysis closely following the work in [217]. The basic idea is as follows:

- i. We consider a simplified $(1+1)$ -dimensional version of the classical problem (see [Figure 6.2](#)) in which, by virtue of the spacetime dimensionality and geometry of the CTC-wormhole, there are *only* two possible classical paths.
- ii. This problem is mapped to a quantum circuit representation and we use a vacuum state to allow the particles to be present or absent from particular paths.
- iii. An internal degree of freedom that operates like a clock is included into the particle's quantum description, meaning that a measurement of the clock's final state can reveal its path's proper time and therefore which of the two classical alternatives was realized.

It is emphasized that the scenario presented here being effectively $(1+1)$ -dimensional [point (i)] is not a characteristic specified by Friedman et al. in their original study [87], but is part of the modelling of the paradox. In addition, the model is based upon two distinct mechanisms. The first of these [point (ii)] involves the introduction of a “vacuum” state: by considering such a state to represent the absence of a billiard ball, we can use it to allow the associated clock to either travel unperturbed (if there is nothing, i.e., a vacuum, in the CTC) or otherwise be scattered into the CTC (if the billiard ball's future self is trapped inside the CTC). The second mechanism [point (iii)] of the model consists of an internal degree of freedom which is incorporated into the billiard ball's quantum description. Using this modification, we can measure the proper time of the model's distinct classical evolutions, which is to say that the billiard ball effectively functions like a clock. This allows us to operationally extract which-way information, i.e., determine which history the billiard ball experienced. This is necessary to give operational meaning to the probabilities associated with the two classical paths, which are indistinguishable by position measurements alone.

It is important to reiterate that it is only the functionality of the clock which makes the two evolutions of the billiard-ball paradox distinguishable. In the Friedman et al. conjecture [87], the use of a semiclassical sum-over-histories approach is only meaningful when the evolutions α and β are treated distinctly *a priori*. This enables one to assign evolution probabilities without the need for conventional which-way information. It should be stressed that Friedman et al. do not provide an operational prescription on how to associate paths with probability, which is the reason why we employ a quantum clock.

One of the open questions in the study of the quantum mechanics of time travel that can be investigated using this model is whether a quantum description does indeed address the issue of evolution indeterminism. As we shall see, a parametrization of the solutions in the D-CTCs description is observed. Although usually interpreted as solution multiplicity, this arises naturally as a choice of initial state in the CTC. Alternatively, in P-CTCs, only one quantum state is offered by the prescription as a solution to the paradox. This state takes the form of a pure superposition of the clock having evolved and not evolved on the CTC during its history through the time machine region. A focus of these P-CTC findings is how this superposition lends credence to the semiclassical billiard-ball conjecture of Friedman et al. in [87]. The well-known problem of how P-CTCs place constraints on the initial data (where such restrictions depend on the future of the time-travelling state), and how this limits the actions that one is able to perform in the billiard-ball interaction, is also discussed.

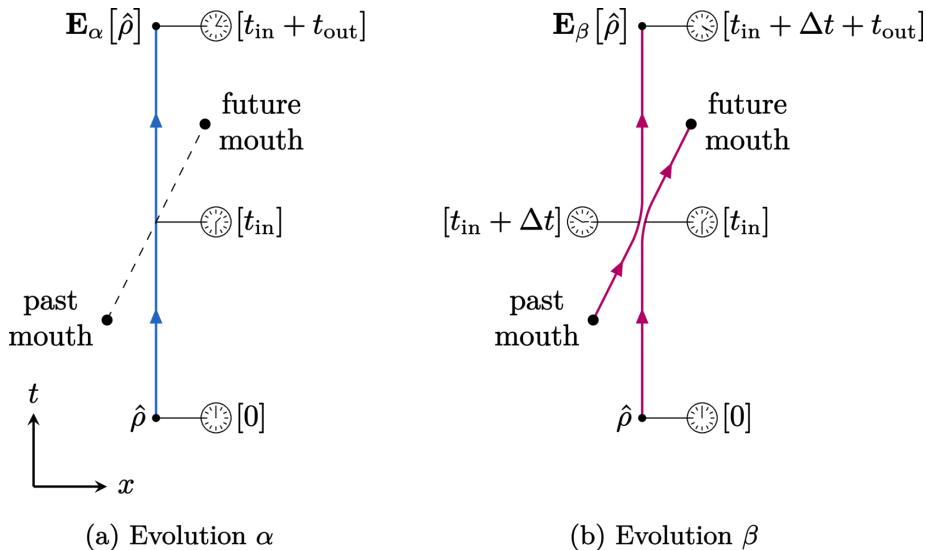


Figure 16.3: A simplified, $(1+1)$ -dimensional version of the billiard-ball paradox (Figure 6.2). This diagram visualizes the basic idea of the model under study in the reference frame of the clock. (a) The history where the clock travels through the CTC-wormhole region with no interaction; (b) the history of a clock which elastically interacts with itself. The times as measured by the clock at specific points along the histories are given in brackets next to their analogue clock symbols. The clocks begin at time $t = 0$ and, due to their initial velocity, measure the proper times $t_\alpha = t_{\text{in}} + t_{\text{out}}$ and $t_\beta = t_{\text{in}} + \Delta t + t_{\text{out}} = t_\alpha + \Delta t$ for evolutions α and β , respectively. This means that we denote the duration of the segment on the CTC in evolution β to be Δt , while the inbound and outbound times to and from the wormhole axis (dashed line) are t_{in} and t_{out} , respectively. It is important to note that the final position bears no information about the path the ball has taken.

16.3.1.1 Model

A quantum “billiard ball” can be modelled simply as a quantum particle, described by a qudit state, that moves though a $(1 + 1)$ -dimensional spacetime. For the billiard-ball paradox, such a spacetime consists of a time-machine wormhole, the mouths of which appear and disappear in such a way as to allow only the two desired paths (see Figure 16.3). We use the quantum billiard ball’s qudit degree of freedom to encode a “clock”, as described in the subsequent section. It is important to note that the particle’s position degree of freedom begins as a semiclassical wave packet, with negligible probability that it falls into the time machine by its own free evolution. This is a significant point, as it allows us to justify both dropping position from the particle’s description and the consideration of only the two trajectories α and β . It also connects with the WKB approximation mentioned by Friedman et al., who propose to apply the sum-over-histories only to the semiclassical trajectories (thus excluding trajectories which pass into the time machine without collision).

For simplicity and rigour, we work in $(1+1)$ -dimensions, which means the only interaction that can occur between the billiard-ball clock's future and past selves is a complete exchange of momentum. This is a useful characteristic, as a quantum SWAP gate between two qudit systems perfectly replicates this action within the quantum circuit framework. Additionally, to allow the billiard-ball paradox for the clock to function as intended, we also require the time machine to be dynamic. This means that, at least in the geometrical picture, the mouths of the wormhole appear and disappear at exactly the right points in the spacetime to allow the clock to evolve in two distinct ways within the chronology-violating region.

It is also important to note that in practice, the methods discussed in *Wormhole-based time machines* (page 59) are likely incapable of constructing a time machine precisely to this required specification. However, as the physical origin of this time machine is not relevant to the quantum mechanics of the problem at hand, we simply do not concern ourselves with such details. We therefore proceed under the assumption that CTCs exactly satisfying our conditions may not strictly be physically realizable, but it is nevertheless the consequences of their existence, not their existence itself, in which we are interested. In addition to this, it is important to note that, due to our abstracting the geometry of the problem away, the findings from this quantum model are independent of the specific kind of CTC (such as those in, e.g.,

the wormhole-based time machine, the Gödel universe, the Kerr solution, etc.) employed to facilitate the requisite time-travel mechanism. In the billiard-ball paradox's original, motivating formulation (*Time-travelling billiard balls* (page 61)), the CTCs are generated specifically by a wormhole-based time machine, but this fact has no consequence in our abstract, quantum formulation of the paradox—the time-travel trajectories in the quantum picture are CTCs in perhaps the purest sense of the concept—and so any CTC possessing the necessary characteristics will suffice.

In any case, the resulting non-collisional evolution α from this specification of the time machine is trivial: the particle remains stationary as the past wormhole mouth appears and disappears behind it, while the future mouth later appears and disappears in front of the particle. Nothing passes in or out of the time machine. On the other hand, in the collisional evolution β , a moving particle emerges from the appearing past mouth and strikes the stationary particle, transferring all of its momentum to it. The struck particle then travels into the future mouth that appears directly in front of it, while the other particle becomes stationary and remains so in place of its collision partner. While these trajectories are classical, the particle can be in superpositions of being absent or present on either path. It is stressed that the particle's position degree of freedom factors out of the evolution because the two classical trajectories coincide in final position and velocity. Therefore, the only relevant degrees of freedom are the internal states of the clock.

It is also important to note that the incoming past and outgoing future momenta of the billiard-ball clock must exactly match. Assuming a perfectly elastic collision, the combination of the dynamical time machine and the law of conservation of momentum collectively constrains the velocity of the billiard-ball clock on the CTC to a single value (determined by both the geometry of the wormhole and mass of the billiard ball). As a result, there is no need to restrict the paths of the clock (e.g., with waveguides) or its initial velocity because the two distinct, desired histories of our model arise naturally as the *only* two paths through the spacetime. With this, we can represent the two evolutions of our clock in its reference frame through the chronology-violating region as per [Figure 16.3](#), where we denote the proper travel time of the clock through the wormhole to be exactly Δt .

16.3.1.2 Quantum clocks

For our analysis, we employ the d -level, pure, equiprobabilistic quantum *clock* state

$$|\theta\rangle = \frac{1}{\sqrt{d}} \sum_{n=1}^d |n\rangle. \quad (16.18)$$

Here, the number states $\{|n\rangle\}_{n=1}^d$, which obey orthonormality via the condition $\langle m|n\rangle = \delta_{nm}$, collectively form a basis for the d -dimensional Hilbert space in which the clock resides. With this, let us define the clock Hamiltonian as

$$\hat{H}_c = \sum_{n=1}^d E_n |n\rangle\langle n| \quad (16.19)$$

where E_n is the energy of the n -th eigenstate $|n\rangle$. Time evolution of a state over the times $t' \rightarrow t''$ is then generated by the time-evolution unitary, which in its standard form may be written

$$\hat{R}(t''; t') \equiv \hat{R}(t'' - t') = e^{-i\hat{H}_c(t'' - t')/\hbar}. \quad (16.20)$$

This is written using the standard notation for the rotation operator \hat{R} because the transformation of a clock state produced by “time evolution” is simply a standard rotation operation. Given this form, the evolution of an initial clock state $|\theta(t)\rangle$ (at time t) to a later state $|\theta(t + \Delta t)\rangle$ (at time $t + \Delta t$) is simply

$$|\theta(t + \Delta t)\rangle = \hat{R}(\Delta t)|\theta(t)\rangle = e^{-i\hat{H}_c\Delta t/\hbar}|\theta(t)\rangle. \quad (16.21)$$

Expanding (16.20) via a Taylor series and subsequently using the Hamiltonian expansion (16.19) allows us to express an evolved state $|\theta(t)\rangle$ in terms of its constituent basis states $\{|n\rangle\}_{n=1}^d$ as

$$|\theta(t)\rangle = \frac{1}{\sqrt{d}} \sum_{n=1}^d e^{-iE_n t/\hbar} |n\rangle. \quad (16.22)$$

The overlap between two states at different stages of time evolution is easily calculable as

$$\langle \theta(t) | \theta(t + \Delta t) \rangle = \frac{1}{d} \sum_{n=1}^d e^{-iE_n \Delta t / \hbar}, \quad (16.23)$$

which quantifies the distinguishability (i.e., coherence) between the two clock states. Now, given our freedom of choice in specifying the energies without losing generality or functionality of the clock, we may choose them to be equally spaced such that

$$\Delta E = E_n - E_{n-1}. \quad (16.24)$$

Accordingly, the n -th energy E_n can be expressed in terms of the ground state E_1 and the spacing ΔE as

$$E_n = E_1 + (n - 1)\Delta E. \quad (16.25)$$

From this, we introduce the orthogonalization time of our d -level clock, defined as

$$t_\perp = \frac{2\pi\hbar}{d\Delta E}, \quad (16.26)$$

which represents the minimal time needed for a clock to evolve (under the equally spaced Hamiltonian (16.19)) into an orthogonal one. A system with finite t_\perp can be thought of as a clock which “ticks” at a rate proportional to t_\perp^{-1} . Now, using the form (16.26) and the energy level relation (16.25) allows us to write the clock overlap (16.23) as

$$\langle \theta(t) | \theta(t + \Delta t) \rangle = \frac{e^{-iE_1 \Delta t / \hbar}}{d} \sum_{n=1}^d \exp \left[-2\pi i \frac{n-1}{d} \frac{\Delta t}{t_\perp} \right]. \quad (16.27)$$

A point to note is that these clocks, when composed of equally spaced energy eigenstates as imposed by (16.25), are dependent on only two parameters: the time t of their evolution and the energy E_1 of their ground state $|1\rangle$. Mathematically, this can be shown by using (16.25) and (16.26) to express (16.22) as

$$|\theta(t)\rangle = \frac{e^{-iE_1 t / \hbar}}{\sqrt{d}} \sum_{n=1}^d \exp \left[-2\pi i \frac{n-1}{d} \frac{t}{t_\perp} \right] |n\rangle. \quad (16.28)$$

In any case, by the series identity

$$\sum_{n=1}^d r^{n-1} = \frac{1 - r^d}{1 - r}, \quad r \neq 1, \quad (16.29)$$

we may write (16.27) as

$$\langle \theta(t) | \theta(t + \Delta t) \rangle = \frac{e^{-iE_1 \Delta t / \hbar}}{d} \frac{1 - \exp \left[-2\pi i \frac{\Delta t}{t_\perp} \right]}{1 - \exp \left[-2\pi i \frac{1}{d} \frac{\Delta t}{t_\perp} \right]}. \quad (16.30)$$

Evidently, in the case that the evolution time difference is equal to the orthogonalization time, i.e., $\Delta t = t_\perp$, then from (16.30) it is easy to see that the clock overlap vanishes, i.e.

$$\langle \theta(t) | \theta(t + \Delta t) \rangle|_{\Delta t = t_\perp} = 0. \quad (16.31)$$

The case of $\Delta t = t_\perp$ can thus be interpreted as when the clock’s “resolution” exactly matches the time difference between the relevant states. This is the key mechanism with which one can investigate temporal differences between the multiple trajectories (such as the two paths in the simple billiard-ball paradox) that are generated via the indeterminism present in spacetimes containing CTCs. Note that such modelling of an internal clock to track a quantum particle’s proper time was first described in [280].

As a final point, note that the simple form of the clock state and its associated Hamiltonian are only convenient choices—the results we obtain from the model of the billiard-ball paradox are completely

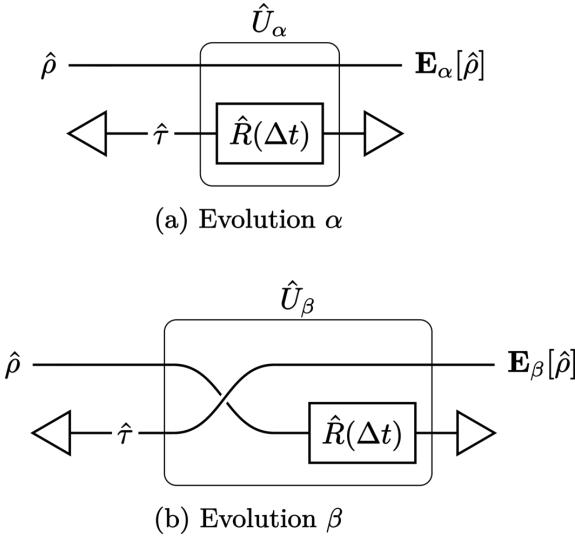
independent of them (provided the initial clock is not an energy eigenstate but is a superposition of different energies). In particular, given that any quantum system can be decomposed into an orthogonal energy basis, then the use of such a basis poses no restrictions on any findings. The use of uniform $1/\sqrt{d}$ weights in the initial state also merely provides simplicity compared to that of general amplitudes $\{c_n\}_{n=1}^d$. In the same vein, the fact that the energy levels are chosen to be equally spaced means that the orthogonalization time is the same between orthogonal states. Non-equally spaced energies on the other hand would only prevent the simplification of the D-CTC solutions. Indeed, the only consequential assumption which we make of the clock system is that it is finite-dimensional, a fact that can be justified by assuming that the clock states must belong to some set of bound states.

16.3.1.3 Quantum circuit model of the billiard-ball paradox

We now turn our attention to constructing the quantum circuit model of the simplified billiard-ball paradox. For simplicity, we choose the initial (unevolved) clock state,

$$\hat{\rho} = |\theta(0)\rangle\langle\theta(0)|, \quad (16.32)$$

to be the CR input along the trajectories illustrated in [Figure 16.3](#). Given these trajectories, it is easy to construct quantum circuit formulations for the clock states, and such circuits appear in [Figure 16.4](#).



[Figure 16.4](#): Quantum circuit models of the two trajectories of [Figure 16.3](#) under study.

Here, the elastic self-collision of the clock between its past and future selves may be represented by the SWAP gate,

$$\hat{S} = \sum_{j,k=1}^d |j\rangle\langle k|_{\text{CR}} \otimes |k\rangle\langle j|_{\text{CV}}. \quad (16.33)$$

In effect, this exchange of the CV (trapped CTC) and CR (incoming system) quantum states mimics the momentum-exchange collision interaction between the billiard ball's past and future selves [which necessarily occurs in $(1+1)$ dimensions]. (Note that here, the basis states correspond to those from which the particle's wave function is constructed. This means that, in the case of a quantum clock, these basis states represent the number states of the clock's Hilbert space.) This is because the complete transference of momentum between two particles (here, the past and future versions of the billiard ball) is necessarily an exchange of quantum clock state, provided the associated particles are otherwise identical. To clarify, in a particle's quantum description, the clock degree of freedom is the identifying attribute, while momentum performs the same function in its classical description. As such, the two necessarily correspond such that when a pair of classically indistinguishable particles trade momentum, they exchange their clocks, i.e., $\hat{S}|\theta(t_1)\rangle \otimes |\theta(t_2)\rangle = |\theta(t_2)\rangle \otimes |\theta(t_1)\rangle$.

Additionally, for simplicity, the rotations of the clock on the path sections outside of the CTC (on the CV system between the time machine) have been neglected. This is valid because these time evolutions

(namely the inbound t_{in} and outbound t_{out} times) are experienced by both paths, and so contribute the same to the clock's total time evolution. This means that a clock which does not interact with the CTC will not evolve, while one which does interact will evolve (by the CTC "duration" Δt).

Next, in order to combine the two separate subcircuits of [Figure 16.4](#) into a single circuit, we introduce a "vacuum" state $|0\rangle$ (orthonormal to the existing number states) into both the CR and CV Hilbert spaces. Of the many consequences of doing so, perhaps the most significant is that, because the clock's subspace has d dimensions, the total space now has $d + 1$ dimensions.

In order to model the paradox faithfully, we mandate that the vacuum and clock cannot "collide" with each other. This effect will be enforced by modifying the SWAP gate to the form

$$\hat{\mathbf{S}} = \hat{\mathbf{I}}_{\text{CR}} \otimes |0\rangle\langle 0|_{\text{CV}} + |0\rangle\langle 0|_{\text{CR}} \otimes \hat{\mathbf{I}}_{\text{CV}} - |0\rangle\langle 0|_{\text{CR}} \otimes |0\rangle\langle 0|_{\text{CV}} + \hat{S} \quad (16.34)$$

where $\hat{\mathbf{I}} = \hat{I} + |0\rangle\langle 0|$ is the identity matrix in a vacuum-inclusive Hilbert space. This altered SWAP simply excludes the vacuum from swapping with the non-vacuous states, e.g.,

$$\begin{aligned} \hat{\mathbf{S}}|0\rangle \otimes |0\rangle &= |0\rangle \otimes |0\rangle, \\ \hat{\mathbf{S}}|\theta\rangle \otimes |0\rangle &= |\theta\rangle \otimes |0\rangle, \\ \hat{\mathbf{S}}|0\rangle \otimes |\theta\rangle &= |0\rangle \otimes |\theta\rangle, \\ \hat{\mathbf{S}}|\theta(t_1)\rangle \otimes |\theta(t_2)\rangle &= |\theta(t_2)\rangle \otimes |\theta(t_1)\rangle. \end{aligned} \quad (16.35)$$

In this interpretation, the state $|0\rangle$ represents the physical absence of a clock $|\theta\rangle$, hence the "vacuum" nomenclature.

In the associated extended Hilbert space, we write the vacuum-inclusive time-evolution operator as

$$\hat{\mathbf{R}}(t'' - t') = |0\rangle\langle 0| + \hat{R}(t'' - t'). \quad (16.36)$$

This form describes the special case where, without loss of generality in the model's results, the vacuum has vanishing energy such that its time-evolution phase coefficient is unity, i.e.,

$$\langle 0|\hat{\mathbf{R}}|0\rangle = 1. \quad (16.37)$$

With this in mind, the corresponding vacuum-modified unitary operator is

$$\hat{U} = [\hat{\mathbf{I}}_{\text{CR}} \otimes \hat{\mathbf{R}}_{\text{CV}}(\Delta t)]\hat{\mathbf{S}}. \quad (16.38)$$

Under this, the input state may either self-consistently interact with the CTC, or it may pass straight through the circuit, thus allowing it to evolve through the circuit in the two paths of [Figure 16.3](#).

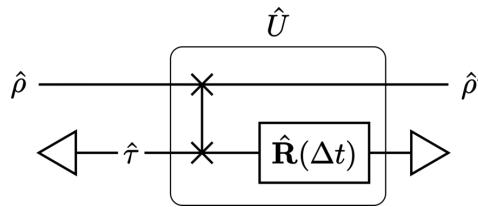


Figure 16.5: A quantum model of the (simplified) billiard-ball paradox. Here, the SWAP gate is a modified $|0\rangle$ -excluding variant as defined in [\(16.34\)](#).

16.3.2 Solutions

The D-CTC solutions can be calculated to be

$$\begin{aligned} \hat{\rho}_{\mathbf{D}} &= g|\theta(0)\rangle\langle\theta(0)| + (1-g)|\theta(\Delta t)\rangle\langle\theta(\Delta t)| \\ \hat{\tau}_{\mathbf{D}} &= g|0\rangle\langle 0| + (1-g)|\theta(\Delta t)\rangle\langle\theta(\Delta t)|. \end{aligned} \quad (16.39)$$

Here, $0 \leq g \leq 1$ is the multiplicity parameter that assumes a value of $g = \frac{1}{d+1}$ in the case of Deutsch's rule of maximal entropy.

For P-CTCs on the other hand, we first compute the P-CTC operator,

$$\begin{aligned}\hat{\mathcal{P}} &\equiv \text{tr}_{\text{CV}}[\hat{U}] \\ &= \text{tr}[\hat{\mathbf{R}}(\Delta t)]|0\rangle\langle 0| + \hat{I} + \hat{R}(\Delta t).\end{aligned}\quad (16.40)$$

with which the corresponding solution can be determined to be

$$|\psi_{\mathbf{P}}\rangle = \frac{1}{\sqrt{\mathcal{N}}} [|\theta(0)\rangle + |\theta(\Delta t)\rangle], \quad (16.41)$$

where

$$\begin{aligned}\mathcal{N} &= \text{tr}[\hat{\mathcal{P}}\hat{\rho}\hat{\mathcal{P}}^\dagger], \\ &= 2 + \frac{1}{d} \left(\text{tr}[\hat{R}(\Delta t)] + \text{tr}[\hat{R}^\dagger(\Delta t)] \right).\end{aligned}\quad (16.42)$$

is the normalization factor. Additionally, although it has not been proven to be valid for non-qubit systems, the generalized weak-measurement tomography expression of the P-CTC CV state yields

$$\hat{\tau}_{\mathbf{P}} = \frac{1}{d+1} [|0\rangle\langle 0| + d|\theta(0)\rangle\langle\theta(0)|]. \quad (16.43)$$

Note how this is exactly the maximally entropic version of the D-CTC solution, as predicted by the theory.

16.3.2.1 Qubit clocks

A particularly elucidating special case of the model discussed so far is one in which the clock's subspace is an ordinary qubit space with basis vectors $\{|1\rangle, |2\rangle\}$. Accordingly, an initial (unevolved) clock state appears as

$$|\theta(0)\rangle = \frac{1}{\sqrt{2}} (|1\rangle + |2\rangle). \quad (16.44)$$

By (16.25) and (16.26), the associated clock gate may be written as

$$\hat{R}(\Delta t) = e^{-iE_1\Delta t/\hbar} (|1\rangle\langle 1| + e^{-i\pi\Delta t/t_\perp} |2\rangle\langle 2|), \quad (16.45)$$

so that the orthogonalization time $\Delta t = t_\perp$ transforms the initial clock qubit (16.44) into the orthogonal state

$$|\theta(t_\perp)\rangle = \hat{R}(t_\perp)|\theta(0)\rangle = \frac{e^{-iE_1t_\perp/\hbar}}{\sqrt{2}} (|1\rangle - |2\rangle). \quad (16.46)$$

Importantly, note that qubit clocks only have two mutually orthogonal states, which means that multiple loops of the CTC cannot be distinguished by such clocks.

In this 3-dimensional total Hilbert space (of both the vacuum and the clock), we compute the D-CTC solutions to be

$$\begin{aligned}\hat{\rho}_{\mathbf{D}} &= \frac{1}{2} [|1\rangle\langle 1| + (g + (1-g)e^{i\pi\Delta t/t_\perp})|1\rangle\langle 2| + (g + (1-g)e^{-i\pi\Delta t/t_\perp})|2\rangle\langle 1| + |2\rangle\langle 2|] \\ \hat{\tau}_{\mathbf{D}} &= g|0\rangle\langle 0| + (1-g)\frac{1}{2} [|1\rangle\langle 1| + e^{i\pi\Delta t/t_\perp}|1\rangle\langle 2| + e^{-i\pi\Delta t/t_\perp}|2\rangle\langle 1| + |2\rangle\langle 2|],\end{aligned}\quad (16.47)$$

while the P-CTC solutions are

$$\begin{aligned}|\psi_{\mathbf{P}}\rangle &= \frac{1}{\sqrt{3 + \cos\left(\frac{\Delta t}{t_\perp}\pi\right)}} [|\theta(0)\rangle + |\theta(\Delta t)\rangle] \\ &\propto 2|1\rangle + (1 + e^{-i\pi\Delta t/t_\perp})|2\rangle, \\ \hat{\tau}_{\mathbf{P}} &= \frac{1}{3} [|0\rangle\langle 0| + 2|\theta(0)\rangle\langle\theta(0)|].\end{aligned}\quad (16.48)$$

16.3.2.2 P-CTC restrictions on initial data

One interesting general characteristic of P-CTCs is that they can pose constraints on initial conditions. Due to the renormalization, such constraints depend on the future of the time-travelling state. Here, we explicitly demonstrate the restrictions associated with the P-CTCs description of the circuit.

Given the d -level clock (16.18), there exist exactly d mutually orthogonal, equally spaced states $\{|\theta^{(k)}(t_{\perp})\rangle\}_{k=0}^{d-1}$ of which it can assume. As a consequence, one may express the clock gate (16.20) in terms of the ground-state energy E_1 (defined via (16.25)) as

$$\hat{R}(t'' - t') = e^{-iE_1(t'' - t')/\hbar} \sum_{n=1}^d \exp\left[-2\pi i \frac{n-1}{d} \frac{t'' - t'}{t_{\perp}}\right] |n\rangle\langle n|. \quad (16.49)$$

From this, it is easy to conclude that

$$\hat{R}^d(t_{\perp}) = e^{-idE_1t_{\perp}/\hbar} \hat{I}, \quad (16.50)$$

which simply indicates that d orthogonal rotations of the clock return it to its initial state, up to the global phase $e^{-idE_1t_{\perp}/\hbar}$. Accordingly, a clock $|\theta\rangle$ that orthogonalizes $p \in \mathbb{Z}_{>0}$ times accumulates p such phases. Therefore, judicious choice of the clock's orthogonalization time t_{\perp} , such that it is related to the CTC time delay Δt by

$$\Delta t = pdt_{\perp}, \quad (16.51)$$

means that a clock which completes one journey on the CTC would orthogonally evolve p times. In such a case, the P-CTC reduced operator (16.40) becomes

$$\hat{\mathcal{P}}\Big|_{\Delta t = pdt_{\perp}} = \left(1 + de^{-ipdE_1t_{\perp}/\hbar}\right) |0\rangle\langle 0| + \left(1 + e^{-ipdE_1t_{\perp}/\hbar}\right) \hat{I}. \quad (16.52)$$

Additionally, if we construct the clock so that its ground-state energy E_1 satisfies

$$E_1 = \frac{\pi\hbar}{pd़t_{\perp}} (1 + 2q), \quad q \in \mathbb{Z}_{\geq 0}, \quad (16.53)$$

then the operator (16.52) further reduces to

$$\hat{\mathcal{P}}' = \hat{\mathcal{P}}\Big|_{\substack{\Delta t = pdt_{\perp} \\ E_1 = \frac{\pi\hbar}{pd़t_{\perp}} (1+2q)}} = (1-d) |0\rangle\langle 0|. \quad (16.54)$$

After renormalization, the corresponding P-CTC output is then, of course, simply the vacuum state, i.e., $|0\rangle$. This cannot be valid however, because the initial input state (16.32) did not include any vacuum term. So what happened to the clock?

To answer this question, we can use the vacuum-including entangled state

$$|\psi\rangle_{\text{rec,CR}} = \sqrt{\Omega} |0\rangle_{\text{rec}} \otimes |0\rangle_{\text{CR}} + \sqrt{1-\Omega} |\theta(0)\rangle_{\text{rec}} \otimes |\theta(0)\rangle_{\text{CR}}, \quad 0 \leq \Omega \leq 1, \quad (16.55)$$

as input to our circuit. Here, the state existing in the first system may be interpreted as a record of what we send into the CR system. With this entanglement, we find the P-CTC unnormalized pure-state output, corresponding to our circuit's ordinary operator $\hat{\mathbf{I}}_{\text{rec}} \otimes \hat{\mathcal{P}}_{\text{CR}}$ from (16.40), to be

$$\begin{aligned} (\hat{\mathbf{I}}_{\text{rec}} \otimes \hat{\mathcal{P}}_{\text{CR}}) |\psi\rangle_{\text{rec,CR}} &\propto \sqrt{\Omega} \text{tr}[\hat{\mathbf{R}}(\Delta t)] |0\rangle_{\text{rec}} \otimes |0\rangle_{\text{CR}} \\ &\quad + \sqrt{1-\Omega} |\theta(0)\rangle_{\text{rec}} \otimes [|\theta(0)\rangle_{\text{CR}} + |\theta(\Delta t)\rangle_{\text{CR}}]. \end{aligned} \quad (16.56)$$

The persistence of the entanglement means that when the record indicates that we did not send in a clock, we will not observe a clock in the CR output. Conversely, when we definitely did send in a clock, we must observe a clock exiting the CTC region, which is exactly what one intuitively expects. However, under the conditions (16.51) and (16.53) which yield the operator (16.54), the entanglement of our input state (16.55) breaks,

$$(\hat{\mathbf{I}}_{\text{rec}} \otimes \hat{\mathcal{P}}'_{\text{CR}}) |\psi\rangle_{\text{rec,CR}} \propto |0\rangle_{\text{rec}} \otimes |0\rangle_{\text{CR}}, \quad (16.57)$$

regardless of the value of Ω (provided it is non-zero). Thus, the mere presence of the interaction with the CTC in the future implies that the record subsystem will show that no clock was ever prepared, even though the initial state (16.55) included the possibility that a clock was there (which would show up in the record if the future interaction with the CTC was avoided).

Given the path-integral correspondence of P-CTCs, the conditions (16.51) and (16.53) collectively form a case in which *no* clocks can evolve through the circuit due to destructive interference in the path integral. This is to say that P-CTCs suppress all evolutions of non-vacuous states as a result of the particular future evolution of clock states in our model, thereby posing constraints on the initial state. This issue highlights the well-known problem of antichronological and superluminal influence with P-CTCs [76, 182, 216]. In contrast, the initial data for D-CTCs are never affected by the presence (or absence) of a CTC in the future.

16.3.3 Implementation

This implementation is strictly for qubits. For simplicity, we set $\frac{\Delta t}{t_\perp} \equiv t$.

Listing 16.3: /text/examples/ctcs/billiards.py

```
from qhronology.quantum.states import VectorState
from qhronology.quantum.gates import QuantumGate, Swap, Diagonal
from qhronology.quantum.prescriptions import QuantumCTC, DCTC, PCTC

import sympy as sp
from sympy.physics.quantum import TensorProduct

# Input
clock_state_unevolved = VectorState(spec=[[0, 1, 1]], dim=3, norm=1, label="ψ")
clock_state_evolved = VectorState(spec=[[0, 1, -1]], dim=3, norm=1, label="ψ")

# Gates
I = QuantumGate(
    spec=[[1, 0, 0], [0, 1, 0], [0, 0, 1]], targets=[0], num_systems=1, dim=3
)
zero = QuantumGate(
    spec=[[1, 0, 0], [0, 0, 0], [0, 0, 0]], targets=[0], num_systems=1, dim=3
)
S = Swap(targets=[0, 1], num_systems=2, dim=3)
IR = Diagonal(
    entries={2: "-pi*t"}, exponentiation=True,
    symbols={"t": {"real": True, "positive": True}},
    targets=[1],
    num_systems=2,
    dim=3,
    label="R",
)

# Construct SWAP gate in clock subspace
S = sp.Matrix(9, 9, lambda i, j: 0 if i % 3 == 0 or j % 3 == 0 else S.matrix[i, j])

# Construct vacuum-excluding SWAP gate
S_vacuum = QuantumGate(
    spec=(
        TensorProduct(I.matrix, zero.matrix)
        + TensorProduct(zero.matrix, I.matrix)
        - TensorProduct(zero.matrix, zero.matrix)
        + S
    )
)
```

(continues on next page)

(continued from previous page)

```

),
targets=[0, 1],
num_systems=2,
dim=3,
family="SWAP",
)

# CTC
billiards = QuantumCTC(
    inputs=[clock_state_unevolved], gates=[S_vacuum, IR], systems_respecting=[0]
)
billiards.diagram()

# Output
# D-CTCs
billiards_DCTC = DCTC(circuit=billiards)
billiards_DCTC_respecting = billiards_DCTC.state_respecting(norm=False, label="ρ_D")
billiards_DCTC_violating = billiards_DCTC.state_violating(norm=False, label="τ_D")
billiards_DCTC_respecting.simplify()
billiards_DCTC_respecting.apply(sp.factor)

# P-CTCs
billiards_PCTC = PCTC(circuit=billiards)
billiards_PCTC_respecting = billiards_PCTC.state_respecting(norm=False, label="ψ_P")
billiards_PCTC_violating = billiards_PCTC.state_violating(norm=False, label="τ_P")
billiards_PCTC_respecting.normalize(1)

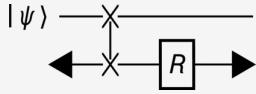
# Results
billiards_DCTC_respecting.print()
billiards_DCTC_violating.print()
billiards_PCTC_respecting.print()
billiards_PCTC_violating.print()

```

16.3.4 Output

16.3.4.1 Diagram

```
>>> billiards.diagram()
```



16.3.4.2 States

```
>>> billiards_DCTC_respecting.print()
ρ_D = 1/2|1⟩⟨1| + -(g*exp(I*pi*t) - g - exp(I*pi*t))/2|1⟩⟨2| + (g*exp(I*pi*t) - g +_
→1)*exp(-I*pi*t)/2|2⟩⟨1| + 1/2|2⟩⟨2|
```

```
>>> billiards_DCTC_violating.print()
τ_D = g|0⟩⟨0| + (1/2 - g/2)|1⟩⟨1| + (1 - g)*exp(I*pi*t)/2|1⟩⟨2| + (1 - g)*exp(-I*pi*t)/
→2|2⟩⟨1| + (1/2 - g/2)|2⟩⟨2|
```

```
>>> billiards_PCTC_respecting.print()
 $|\psi_P\rangle = \sqrt{2} * \sqrt{1 / (\cos(\pi * t) + 3)} |1\rangle + \sqrt{2} * (1 + \exp(-I * \pi * t)) * \sqrt{1 / (\cos(\pi * t) + 3)} / 2 |2\rangle$ 
```

```
>>> billiards_PCTC_violating.print()
 $\tau_P = 1/3 |0\rangle\langle 0| + 1/3 |1\rangle\langle 1| + \exp(I * \pi * t) / 3 |1\rangle\langle 2| + \exp(-I * \pi * t) / 3 |2\rangle\langle 1| + 1/3 |2\rangle\langle 2|$ 
```

16.4 Manual P-CTC

16.4.1 Description

This example merely serves as a proof of the P-CTC prescription (for qubits). This is achieved by explicitly performing the Bell state preselection and postselection, and subsequently comparing the output to the prescription's CR map.

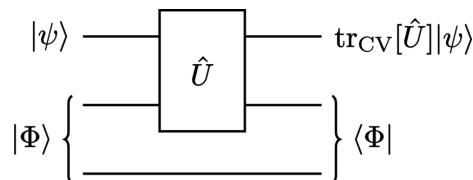


Figure 16.6: The P-CTCs prescription with explicit preselection and postselection.

16.4.2 Implementation

Listing 16.4: /text/examples/ctcs/postselected.py

```

from qchronology.quantum.states import VectorState
from qchronology.quantum.gates import QuantumGate
from qchronology.quantum.circuits import QuantumCircuit

from qchronology.quantum.prescriptions import pctc_respecting

import sympy as sp

# Input
rho = sp.MatrixSymbol("ρ", 2, 2).asMutable()
unitary = sp.MatrixSymbol("U", 4, 4).asMutable()

input_state = VectorState(spec=[("a", [0]), ("b", [1])], label="ψ")
bell_state = VectorState(spec=[(1, [0, 0]), (1, [1, 1])], norm=1, label="Φ")

# Gate
UI = QuantumGate(spec=unitary, targets=[0, 1], num_systems=3, label="U")

# Circuit
postselected_teleportation = QuantumCircuit(
    inputs=[input_state, bell_state], gates=[UI], postselections=[(bell_state, [1, 2])]
)
postselected_teleportation.diagram(pad=(1, 0), sep=(2, 1), force_separation=True)

# Output
# Manual P-CTC
output_state = postselected_teleportation.state(label="ψ")
output_state.apply(sp.factor)

# P-CTC prescription
spec = pctc_respecting(
    input_respecting=input_state,
    gate=unitary,
    systems_respecting=[0],
    systems_violating=[1],
)

```

(continues on next page)

(continued from previous page)

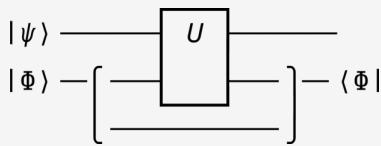
```
pctc_output = VectorState(spec=spec, label=" $\psi_P$ ")
pctc_output.coefficient(sp.Rational(1, 2)) # Manually renormalize
pctc_output.apply(sp.factor)

# Results
input_state.print()
output_state.print()
pctc_output.print()
```

16.4.3 Output

16.4.3.1 Diagram

```
>>> postselected_teleportation.diagram(pad=(1, 0), sep=(1, 1), force_separation=True)
```



16.4.3.2 States

```
>>> input_state.print()
|ψ⟩ = a|0⟩ + b|1⟩
```

```
>>> output_state.print()
|ψ'⟩ = (a*U[0, 0] + a*U[1, 1] + b*U[0, 2] + b*U[1, 3])/2|0⟩ + (a*U[2, 0] + a*U[3, 1] +_
→b*U[2, 2] + b*U[3, 3])/2|1⟩
```

```
>>> pctc_output.print()
|ψ_P⟩ = (a*U[0, 0] + a*U[1, 1] + b*U[0, 2] + b*U[1, 3])/2|0⟩ + (a*U[2, 0] + a*U[3, 1] +_
→b*U[2, 2] + b*U[3, 3])/2|1⟩
```

16.4.3.3 Results

```
>>> output_state.distance(pctc_output)
0
```

16.5 Equivalent-circuit picture for D-CTCs

16.5.1 Description

This algorithm implements the equivalent-circuit picture (ECP, see *Iterative picture: The equivalent circuit* (page 65)) of D-CTCs. In its current form, it is not a particularly useful example, having very general input state and unitary matrix symbolic representations that make any subsequent analysis infeasible. Instead, the algorithm is included simply because it is an interesting demonstration of advanced usage of Qhronology. Note that the larger the number of algebraic symbols (and associated conditions) that are contained within the states and gates, the higher the complexity of the internal calculations, resulting in correspondingly longer computation times.

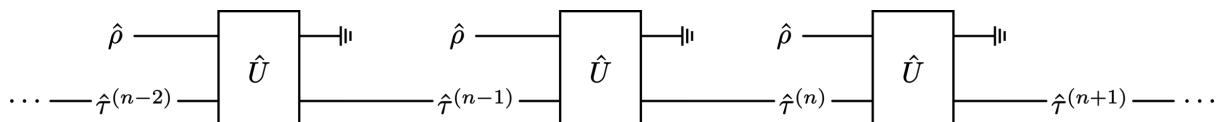


Figure 16.7: The equivalent-circuit picture of a D-CTC.

16.5.2 Implementation

The desired number of iterations can be changed by setting `iterations` to an appropriate positive integer.

Listing 16.5: /text/examples/ctcs/ecp.py

```
from qchronology.quantum.states import MixedState
from qchronology.quantum.gates import QuantumGate
from qchronology.quantum.circuits import QuantumCircuit

import sympy as sp
from sympy.physics.quantum.dagger import Dagger

iterations = 1
dimensionality = 2

# Input
rho = sp.MatrixSymbol("ρ", dimensionality, dimensionality).asMutable()
respecting_state = MixedState(
    spec=rho, dim=dimensionality, conditions=[(sp.trace(rho), 1)], label="ρ"
)
seed_state = MixedState(
    spec=sp.eye(dimensionality), dim=dimensionality, norm=1, label="τ_0"
)

# Gate
unitary = sp.MatrixSymbol("U", dimensionality**2, dimensionality**2).asMutable()
U = QuantumGate(spec=unitary, targets=[0, 1], num_systems=2, dim=dimensionality)

# Construct conditions
conditions_respecting = [(sp.trace(rho), 1)]
conditions_unitary = [
    ((unitary * Dagger(unitary))[n], (sp.eye(dimensionality**2))[n])
    for n in range(0, len(unitary))
]
conditions = conditions_respecting + conditions_unitary
```

(continues on next page)

(continued from previous page)

```

violating_state = seed_state
for n in range(1, iterations + 1):
    iteration = QuantumCircuit(
        inputs=[respecting_state, violating_state],
        gates=[U],
        conditions=conditions,
        traces=[0],
    )
    iteration.diagram(pad=(1, 0), sep=(0, 1), style="unicode")
    violating_state = iteration.state(label=f"\tau_{n}")

# Output
final_state = violating_state

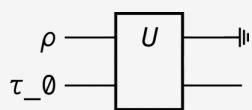
# Results
seed_state.print()

```

16.5.3 Output

16.5.3.1 Diagram

```
>>> iteration.diagram()
```



16.5.3.2 States

```
>>> seed_state.print()
\tau_0 = 1/2|0\rangle\langle 0| + 1/2|1\rangle\langle 1|
```

Part IV

Appendix

List of symbols

Table 17.1: List of symbols

Symbol	Description
i	imaginary unit
e	Euler's number ($\approx 2.718\dots$)
\mathbb{R}	set of real numbers
\mathbb{Z}	set of integers
\mathbb{N}	set of natural numbers
\mathbb{C}	set of complex numbers
\mathcal{V}	vector space
\oplus	addition in modular arithmetic ($x \oplus y \equiv x + y \pmod{m}$ for a modulus $m \in \mathbb{Z}^+$)
\ominus	subtraction in modular arithmetic ($x \ominus y \equiv x - y \pmod{m}$ for a modulus $m \in \mathbb{Z}^+$)
\otimes	tensor product (also known as the Kronecker product in the context of matrices)
\mathcal{H}	Hilbert space
$\langle \cdot, \cdot \rangle$	inner product (the Mathematician's notation)
$\langle \cdot \cdot \rangle$	inner product (the Physicist's notation, "bra-ket")
$\langle \cdot \rangle$	expected value
ℓ^p	space of sequences with converging p -norm
L^p	space of functions with converging p -norm
$\mathcal{P}(\mathcal{H})$	set of pure quantum states on \mathcal{H}
$\mathcal{D}(\mathcal{H})$	set of mixed quantum states (normalized, Hermitian, density operators) on \mathcal{H}
$\mathcal{L}(\mathcal{H})$	set of linear operators on \mathcal{H}
$\dim(\cdot)$	dimension
$\text{tr}[\cdot]$	trace
$\gamma(\cdot)$	purity
$D(\cdot, \cdot)$	trace distance
$F(\cdot, \cdot)$	fidelity
$S(\cdot)$	von Neumann entropy
$S(\cdot \parallel \cdot)$	relative entropy
$I(\cdot : \cdot)$	mutual information
$S[\mathbf{x}(t)]$	classical action over a path $\mathbf{x}(t)$
$K(\mathbf{x}'', t''; \mathbf{x}', t')$	(non-relativistic) quantum propagator from (t', \mathbf{x}') to (t'', \mathbf{x}'')
$D\mathbf{x}(t)$	path-integral formulation differential
t	time
\mathbf{x}	position vector
\mathbf{p}	momentum vector
\hat{H}	Hamiltonian operator
\hat{L}	Lagrangian operator
\hat{T}	kinetic energy operator
\hat{V}	potential energy operator
\mathcal{W}^\pm	wormhole mouths ($\mathcal{W}^+ =$ future, $\mathcal{W}^- =$ past)
$\mathbb{P}(\cdot)$	probability density of a continuous function
$\mathbb{E}(\cdot)$	expected value
$ \cdot\rangle$	quantum vector state (a "ket")
$\langle \cdot $	quantum covector state (a "bra")
$ i\rangle, j\rangle, k\rangle, n\rangle, \dots$	number state (e.g., $ 0\rangle, 1\rangle, 2\rangle, \dots$)
$ \Phi\rangle$	maximally entangled state

continues on next page

Table 17.1 – continued from previous page

Symbol	Description
\hat{I}_d	$d \times d$ identity operator, (unnormalized) maximally mixed state
$\hat{\rho}, \hat{\tau}, \hat{\omega}, \dots$	density operators (lowercase Greek letters)
$\hat{A}, \hat{B}, \hat{C}, \hat{U}, \hat{Q}, \dots$	linear operators (uppercase Latin letters)
$\hat{R}(\theta)$	rotation gate (with angle parameter θ)
$\hat{P}(\omega)$	phase gate (with phase parameter ω)
$\hat{S}(p)$	SWAP gate (with power parameter p)
$\hat{\Sigma}(n)$	SUM gate (with shift parameter n)
\hat{H}	Hadamard gate
\hat{F}	quantum Fourier transform gate
$\hat{\mathbf{S}}, \hat{\mathbf{I}}, \hat{\mathbf{R}}$	vacuum-adapted ($ 0\rangle$ -inclusive) gate variants
$C^\alpha \bar{C}^\beta \hat{U}^\gamma$	controlled-anticontrolled-unitary gate (modes: α = control, β = anticontrol, γ = unitary)
$\hat{\sigma}_\mu$	Pauli matrices (with identity $\hat{\sigma}_0 = \hat{I}$)
$\hat{\lambda}_\mu$	Gell-Mann matrices (with identity $\hat{\lambda}_0 = \hat{I}$)
$\mathbf{D}_{\hat{U}}[\cdot, \cdot]$	D-CTCs CR map
$\mathbf{d}_{\hat{U}}[\cdot, \cdot]$	D-CTCs CV map
$\mathbf{P}_{\hat{U}}[\cdot]$	P-CTCs CR map
$\mathbf{p}_{\hat{U}}[\cdot]$	P-CTCs CV map
$\mathbf{T}_{\hat{U}}[\cdot]$	T-CTCs CR map
$\mathbf{t}_{\hat{U}}[\cdot]$	T-CTCs CV map
$\hat{\mathcal{P}}$	P-CTC operator
$\mathcal{D}(\cdot)$	decoherence channel
$\mathcal{N}(\cdot)$	depolarization channel
\hat{K}	Kraus operator
\mathbb{K}	set of Kraus operators
p_i	probability (density) of a discrete state (labelled with i)
\mathbb{P}	Probability distribution (set of probabilities of discrete states)
\mathbb{U}	set of unitary operators
\mathbb{M}	POVM (positive operator-valued measure) (set of positive operators)

List of abbreviations

Table 18.1: List of symbols

Abbreviation	Description
PVM	projection-valued measure
POVM	positive operator-valued measure
PDF	probability density function
TDSE	time-dependent Schrödinger equation
CTC	closed timelike curve
CTG	closed timelike geodesic
CR	chronology-respecting
CV	chronology-violating
D-CTC	Deutschian closed timelike curve
P-CTC	postselected-teleportation closed timelike curve
T-CTC	transition-probabilities closed timelike curve
ECP	equivalent-circuit picture

Bibliography

- [1] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, Cambridge University Press, Cambridge, 2010.
- [2] E. Zeidler, *Quantum Field Theory I: Basics in Mathematics and Physics*, Springer, Berlin, Heidelberg, 2006.
- [3] J. Audretsch, *Entangled Systems: New Directions in Quantum Physics*, WILEY-VCH, February 2007.
- [4] J. Dimock, *Quantum Mechanics and Quantum Field Theory: A Mathematical Primer*, Cambridge University Press, Cambridge, 2011.
- [5] R. Shankar, *Principles of Quantum Mechanics*, Springer New York, 2nd edition, December 2012.
- [6] M. Ohya and I. Volovich, *Mathematical Foundations of Quantum Information and Computation and Its Applications to Nano- and Bio-systems*, Theoretical and Mathematical Physics, Springer Dordrecht, January 2011.
- [7] L. E. Ballentine, *Quantum Mechanics: A Modern Development*, World Scientific, 2nd edition, November 2014.
- [8] A. S. Holevo, *Quantum Systems, Channels, Information: A Mathematical Introduction*, De Gruyter, December 2012.
- [9] J. Watrous, *The Theory of Quantum Information*, Cambridge University Press, Cambridge, 2018.
- [10] E. Prugovečki, *Quantum Mechanics in Hilbert Space*, volume 92 of Pure and applied mathematics, a series of monographs and textbooks, Academic Press, 2nd edition, August 1982.
- [11] F. Hiai and D. Petz, *Introduction to Matrix Analysis and Applications*, Universitext, Springer Cham, February 2014.
- [12] P. A. M. Dirac, *The Principles of Quantum Mechanics*, Clarendon Press, Oxford, 4th edition, February 1982.
- [13] M. M. Wilde, *Quantum Information Theory*, Cambridge University Press, Cambridge, 2nd edition, 2017.
- [14] I. Bengtsson and K. Życzkowski, *Geometry of Quantum States: An Introduction to Quantum Entanglement*, Cambridge University Press, Cambridge, 2nd edition, 2017.
- [15] C. P. Williams, *Explorations in Quantum Computing*, Springer, London, 2nd edition, December 2010.
- [16] M. A. Schlosshauer, *Decoherence and the Quantum-to-Classical Transition*, The Frontiers Collection, Springer, Berlin ; London, 2007.
- [17] V. Moret-Bonillo, *Adventures in Computer Science: From Classical Bits to Quantum Bits*, Springer International Publishing, Cham, 2017.
- [18] M. A. Serrano, R. Pérez-Castillo, and M. Piattini (editors), *Quantum Software Engineering*, Springer International Publishing, Cham, 2022.
- [19] N. S. Yanofsky and M. A. Mannucci, *Quantum Computing for Computer Scientists*, Cambridge University Press, Cambridge, 2008.
- [20] R. T. Thew, K. Nemoto, A. G. White, and W. J. Munro, “Qudit quantum-state tomography”, *Physical Review A*, 66(1):012303, July 2002. <https://link.aps.org/doi/10.1103/PhysRevA.66.012303>, doi:10.1103/PhysRevA.66.012303

- [21] A. J. Baldwin and J. A. Jones, “Efficiently computing the Uhlmann fidelity for density matrices”, *Physical Review A*, 107(1):012427, January 2023. <https://link.aps.org/doi/10.1103/PhysRevA.107.012427>, doi:10.1103/PhysRevA.107.012427
- [22] A. Müller, “A Simplified Expression for Quantum Fidelity”, October 2023. <http://arxiv.org/abs/2309.10565>, doi:10.48550/arXiv.2309.10565
- [23] J.-M. A. Allen, “Treating time travel quantum mechanically”, *Physical Review A*, 90(4):042107, October 2014. <https://link.aps.org/doi/10.1103/PhysRevA.90.042107>, doi:10.1103/PhysRevA.90.042107
- [24] A. Hurwitz, “Über die Erzeugung der Invarianten durch Integration”, *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1897:71–90, 1897. <https://eudml.org/doc/58378>
- [25] K. Zyczkowski and H.-J. Sommers, “Induced measures in the space of mixed quantum states”, *Journal of Physics A: Mathematical and General*, 34(35):7111, August 2001. <https://dx.doi.org/10.1088/0305-4470/34/35/335>, doi:10.1088/0305-4470/34/35/335
- [26] L. H. Ryder, *Quantum Field Theory*, Cambridge University Press, Cambridge, 2nd edition, 1996.
- [27] A. Zee, *Quantum Field Theory in a Nutshell*, Princeton University Press, Princeton, N.J., 2nd edition, February 2010.
- [28] J. S. Townsend, *A Modern Approach to Quantum Mechanics*, University Science Books, Sausalito, California, 2nd edition, February 2012.
- [29] D. J. Griffiths and D. F. Schroeter, *Introduction to Quantum Mechanics*, Cambridge University Press, 3rd edition, August 2018.
- [30] K. Schulten, *Notes on Quantum Mechanics*, CreateSpace Independent Publishing Platform, August 2014.
- [31] W. Dittrich and M. Reuter, *Classical and Quantum Dynamics: From Classical Paths to Path Integrals*, Springer International Publishing, Cham, 2020.
- [32] L. S. Schulman, *Techniques and Applications of Path Integration*, Wiley, New York, 1981.
- [33] H. Kleinert, *Path Integrals in Quantum Mechanics, Statistics, Polymer Physics, and Financial Markets*, World Scientific, 5th edition, 2009.
- [34] R. P. Feynman, “Quantum mechanical computers”, *Foundations of Physics*, 16(6):507–531, June 1986. <https://doi.org/10.1007/BF01886518>, doi:10.1007/BF01886518
- [35] A. Kay, “Tutorial on the Quantikz Package”, 2019. <http://arxiv.org/abs/1809.03842>, doi:10.17637/rh.7000520
- [36] K. Lanczos, “Über eine stationäre Kosmologie im Sinne der Einsteinschen Gravitationstheorie”, *Zeitschrift für Physik*, 21(1):73–110, December 1924. <https://doi.org/10.1007/BF01328251>, doi:10.1007/BF01328251
- [37] T. Lewis, “Some special solutions of the equations of axially symmetric gravitational fields”, *Proc. R. Soc. Lond. A*, 136(829):176–192, May 1932. <http://rspa.royalsocietypublishing.org/content/136/829/176>, doi:10.1098/rspa.1932.0073
- [38] W. J. van Stockum, “The Gravitational Field of a Distribution of Particles Rotating about an Axis of Symmetry”, *Proceedings of the Royal Society of Edinburgh*, 57:135–154, 1938. <https://www.cambridge.org/core/journals/proceedings-of-the-royal-society-of-edinburgh/article/ixthe-gravitational-field-of-a-distribution-of-particles-rotating-about-an-axis-of-symmetry/40E39372658C7031B0C4316A36154F46>, doi:10.1017/S0370164600013699
- [39] A. Papapetrou, “A Static Solution of the Equations of the Gravitational Field for an Arbitrary Charge-Distribution”, *Proceedings of the Royal Irish Academy. Section A: Mathematical and Physical Sciences*, 51:191–204, 1945. <http://www.jstor.org/stable/20488481>
- [40] K. Gödel, “An Example of a New Type of Cosmological Solutions of Einstein's Field Equations of Gravitation”, *Reviews of Modern Physics*, 21(3):447–450, July 1949. <https://link.aps.org/doi/10.1103/RevModPhys.21.447>, doi:10.1103/RevModPhys.21.447

- [41] A. H. Taub, “Empty Space-Times Admitting a Three Parameter Group of Motions”, *Annals of Mathematics*, 53(3):472–490, 1951. <http://www.jstor.org/stable/1969567>, doi:10.2307/1969567
- [42] E. T. Newman, L. A. Tamburino, and T. W. J. Unti, “Empty-Space Generalization of the Schwarzschild Metric”, *Journal of Mathematical Physics*, 4(7):915–923, July 1963. <http://aip.scitation.org/doi/10.1063/1.1704018>, doi:10.1063/1.1704018
- [43] R. P. Kerr, “Gravitational Field of a Spinning Mass as an Example of Algebraically Special Metrics”, *Physical Review Letters*, 11(5):237–238, September 1963. <https://link.aps.org/doi/10.1103/PhysRevLett.11.237>, doi:10.1103/PhysRevLett.11.237
- [44] H. Levy and W. J. Robinson, “The rotating body problem”, *Mathematical Proceedings of the Cambridge Philosophical Society*, 60(2):279–285, April 1964. <https://www.cambridge.org/core/journals/mathematical-proceedings-of-the-cambridge-philosophical-society/article/rotating-body-problem/BFCD29EC2C11E38FD0E2200B658D5456>, doi:10.1017/S0305004100037750
- [45] A. Papapetrou, “Champs gravitationnels stationnaires à symétrie axiale”, *Ann. Inst. H. Poincaré Phys. Théor.*, 4:83–105, 1966. http://www.numdam.org/item/AIHPC_1966__4_2_83_0/
- [46] F. J. Tipler, “Rotating cylinders and the possibility of global causality violation”, *Physical Review D*, 9(8):2203–2206, April 1974. <https://link.aps.org/doi/10.1103/PhysRevD.9.2203>, doi:10.1103/PhysRevD.9.2203
- [47] I. Damião Soares, “Inhomogeneous rotating universes with closed timelike geodesics of matter”, *Journal of Mathematical Physics*, 21(3):521–525, March 1980. <https://aip.scitation.org/doi/10.1063/1.524437>, doi:10.1063/1.524437
- [48] J. R. Gott, “Closed timelike curves produced by pairs of moving cosmic strings: Exact solutions”, *Physical Review Letters*, 66(9):1126–1129, March 1991. <https://link.aps.org/doi/10.1103/PhysRevLett.66.1126>, doi:10.1103/PhysRevLett.66.1126
- [49] W. B. Bonnor, “An exact, asymptotically flat, vacuum solution of Einstein's equations with closed timelike curves”, *Classical and Quantum Gravity*, 19(23):5951–5957, November 2002. <https://doi.org/10.1088/0264-9381/19/23/305>, doi:10.1088/0264-9381/19/23/305
- [50] W. B. Bonnor and B. R. Steadman, “The double-Kerr solution”, *Classical and Quantum Gravity*, 21(11):2723–2732, May 2004. <https://doi.org/10.1088/0264-9381/21/11/014>, doi:10.1088/0264-9381/21/11/014
- [51] A. Ori, “A Class of Time-Machine Solutions with a Compact Vacuum Core”, *Physical Review Letters*, 95(2):021101, July 2005. <https://link.aps.org/doi/10.1103/PhysRevLett.95.021101>, doi:10.1103/PhysRevLett.95.021101
- [52] W. B. Bonnor and B. R. Steadman, “Exact solutions of the Einstein-Maxwell equations with closed timelike curves”, *General Relativity and Gravitation*, 37(11):1833–1844, November 2005. <https://doi.org/10.1007/s10714-005-0163-3>, doi:10.1007/s10714-005-0163-3
- [53] A. Ori, “Formation of closed timelike curves in a composite vacuum/dust asymptotically flat spacetime”, *Physical Review D*, 76(4):044002, August 2007. <https://link.aps.org/doi/10.1103/PhysRevD.76.044002>, doi:10.1103/PhysRevD.76.044002
- [54] D. Sarma, M. Patgiri, and F. U. Ahmed, “Pure radiation metric with stable closed timelike curves”, *General Relativity and Gravitation*, 46(1):1633, December 2013. <https://doi.org/10.1007/s10714-013-1633-7>, doi:10.1007/s10714-013-1633-7
- [55] B. K. Tippett and D. Tsang, “Traversable acausal retrograde domains in spacetime”, *Classical and Quantum Gravity*, 34(9):095006, March 2017. <https://doi.org/10.1088/1361-6382/aa6549>, doi:10.1088/1361-6382/aa6549
- [56] D. Fermi and L. Pizzocchero, “A time machine for free fall into the past”, *Classical and Quantum Gravity*, 35(16):165003, July 2018. <https://doi.org/10.1088/1361-6382/aace6e>, doi:10.1088/1361-6382/aace6e

- [57] T. C. Ralph and C. Chang, “Spinning up a time machine”, *Physical Review D*, 102(12):124013, December 2020. <https://link.aps.org/doi/10.1103/PhysRevD.102.124013>, doi:10.1103/PhysRevD.102.124013
- [58] H. G. Ellis, “Ether flow through a drainhole: A particle model in general relativity”, *Journal of Mathematical Physics*, 14(1):104–118, January 1973. <http://aip.scitation.org/doi/abs/10.1063/1.1666161>, doi:10.1063/1.1666161
- [59] S. W. Hawking, “Wormholes in spacetime”, *Physical Review D*, 37(4):904–910, February 1988. <https://link.aps.org/doi/10.1103/PhysRevD.37.904>, doi:10.1103/PhysRevD.37.904
- [60] M. S. Morris and K. S. Thorne, “Wormholes in spacetime and their use for interstellar travel: A tool for teaching general relativity”, *American Journal of Physics*, 56(5):395–412, May 1988. <https://aapt.scitation.org/doi/10.1119/1.15620>, doi:10.1119/1.15620
- [61] M. S. Morris, K. S. Thorne, and U. Yurtsever, “Wormholes, Time Machines, and the Weak Energy Condition”, *Physical Review Letters*, 61(13):1446–1449, September 1988. <https://link.aps.org/doi/10.1103/PhysRevLett.61.1446>, doi:10.1103/PhysRevLett.61.1446
- [62] M. Visser, “Traversable wormholes: Some simple examples”, *Physical Review D*, 39(10):3182–3184, May 1989. <https://link.aps.org/doi/10.1103/PhysRevD.39.3182>, doi:10.1103/PhysRevD.39.3182
- [63] M. Visser, “Traversable wormholes from surgically modified Schwarzschild spacetimes”, *Nuclear Physics B*, 328(1):203–212, December 1989. <https://www.sciencedirect.com/science/article/pii/0550321389901004>, doi:10.1016/0550-3213(89)90100-4
- [64] D. H. Coule and K. I. Maeda, “Wormholes with scalar fields”, *Classical and Quantum Gravity*, 7(6):955–963, June 1990. <https://doi.org/10.1088%2F0264-9381%2F7%2F6%2F005>, doi:10.1088/0264-9381/7/6/005
- [65] M. Visser, “Van Vleck determinants: Traversable wormhole spacetimes”, *Physical Review D*, 49(8):3963–3980, April 1994. <https://link.aps.org/doi/10.1103/PhysRevD.49.3963>, doi:10.1103/PhysRevD.49.3963
- [66] M. Visser, *Lorentzian Wormholes: From Einstein to Hawking*, AIP Press, American Institute of Physics, 1995.
- [67] M. Visser, “Traversable wormholes: The Roman ring”, *Physical Review D*, 55(8):5212–5214, April 1997. <https://link.aps.org/doi/10.1103/PhysRevD.55.5212>, doi:10.1103/PhysRevD.55.5212
- [68] A. Carlini, D. H. Coule, and D. M. Solomons, “Euclidean Quantum Wormholes with Scalar Fields”, *International Journal of Modern Physics A*, 12(20):3517–3544, August 1997. <http://www.worldscientific.com/doi/abs/10.1142/S0217751X97001821>, doi:10.1142/S0217751X97001821
- [69] D. Hochberg and M. Visser, “Geometric structure of the generic static traversable wormhole throat”, *Physical Review D*, 56(8):4745–4755, October 1997. <https://link.aps.org/doi/10.1103/PhysRevD.56.4745>, doi:10.1103/PhysRevD.56.4745
- [70] C. Barceló and M. Visser, “Traversable wormholes from massless conformally coupled scalar fields”, *Physics Letters B*, 466(2):127–134, November 1999. <https://www.sciencedirect.com/science/article/pii/S037026939901117X>, doi:10.1016/S0370-2693(99)01117-X
- [71] C. Barceló and M. Visser, “Scalar fields, energy conditions and traversable wormholes”, *Classical and Quantum Gravity*, 17(18):3843–3864, September 2000. <https://doi.org/10.1088/0264-9381/17/18/318>, doi:10.1088/0264-9381/17/18/318
- [72] C. Barceló and M. Visser, “Brane surgery: energy conditions, traversable wormholes, and voids”, *Nuclear Physics B*, 584(1):415–435, September 2000. <https://www.sciencedirect.com/science/article/pii/S0550321300003795>, doi:10.1016/S0550-3213(00)00379-5
- [73] N. Dadhich, S. Kar, S. Mukherjee, and M. Visser, “ $\$R=0\$$ spacetimes and self-dual Lorentzian wormholes”, *Physical Review D*, 65(6):064004, February 2002. <https://link.aps.org/doi/10.1103/PhysRevD.65.064004>, doi:10.1103/PhysRevD.65.064004
- [74] M. Visser, S. Kar, and N. Dadhich, “Traversable Wormholes with Arbitrarily Small Energy Condition Violations”, *Physical Review Letters*, 90(20):201102, May 2003. <https://link.aps.org/doi/10.1103/PhysRevLett.90.201102>, doi:10.1103/PhysRevLett.90.201102

- [75] S. Kar, N. Dadhich, and M. Visser, “Quantifying energy condition violations in traversable wormholes”, *Pramana*, 63(4):859–864, October 2004. <https://doi.org/10.1007/BF02705207>, doi:10.1007/BF02705207
- [76] T. C. Ralph and T. G. Downes, “Relativistic quantum information and time machines”, *Contemporary Physics*, 53(1):1–16, January 2012. <http://www.tandfonline.com/doi/abs/10.1080/00107514.2011.640146>
- [77] N. M. Garcia, F. S. N. Lobo, and M. Visser, “Generic spherically symmetric dynamic thin-shell traversable wormholes in standard general relativity”, *Physical Review D*, 86(4):044026, August 2012. <https://link.aps.org/doi/10.1103/PhysRevD.86.044026>, doi:10.1103/PhysRevD.86.044026
- [78] L. M. Butcher, “Traversable wormholes and classical scalar fields”, *Physical Review D*, 91(12):124031, June 2015. <https://link.aps.org/doi/10.1103/PhysRevD.91.124031>, doi:10.1103/PhysRevD.91.124031
- [79] K. A. Bronnikov, “Scalar fields as sources for wormholes and regular black holes”, *Particles*, 1(1):56–81, December 2018. <https://www.mdpi.com/2571-712X/1/1/5>, doi:10.3390/particles1010005
- [80] A. Simpson, P. Martín-Moruno, and M. Visser, “Vaidya spacetimes, black-bounces, and traversable wormholes”, *Classical and Quantum Gravity*, 36(14):145007, June 2019. <https://doi.org/10.1088/1361-6382/ab28a5>, doi:10.1088/1361-6382/ab28a5
- [81] P. Cañate, J. Sultana, and D. Kazanas, “Ellis wormhole without a phantom scalar field”, *Physical Review D*, 100(6):064007, September 2019. <https://link.aps.org/doi/10.1103/PhysRevD.100.064007>, doi:10.1103/PhysRevD.100.064007
- [82] F. S. N. Lobo, A. Simpson, and M. Visser, “Dynamic thin-shell black-bounce traversable wormholes”, *Physical Review D*, 101(12):124035, June 2020. <https://link.aps.org/doi/10.1103/PhysRevD.101.124035>, doi:10.1103/PhysRevD.101.124035
- [83] T. Berry, F. S. N. Lobo, A. Simpson, and M. Visser, “Thin-shell traversable wormhole crafted from a regular black hole with asymptotically Minkowski core”, *Physical Review D*, 102(6):064054, September 2020. <https://link.aps.org/doi/10.1103/PhysRevD.102.064054>, doi:10.1103/PhysRevD.102.064054
- [84] I. D. Novikov, “Analysis of the operation of a time machine”, *Zhurnal Èksperimental'noi Teoreticheskoi Fiziki*, 95(3):769–776, March 1989. <http://jetp.ras.ru/cgi-bin/e/index/r/95/3/p769?a=list>
- [85] K. S. Thorne, “Do the laws of physics permit closed timelike curves?”, *Annals of the New York Academy of Sciences*, 631(1):182–193, August 1991. <http://nyaspubs.onlinelibrary.wiley.com/doi/abs/10.1111/j.1749-6632.1991.tb52642.x>, doi:10.1111/j.1749-6632.1991.tb52642.x
- [86] A. S. Lossev and I. D. Novikov, “The jinn of the time machine: nontrivial self-consistent solutions”, *Classical and Quantum Gravity*, 9(10):2309, 1992. <https://iopscience.iop.org/article/10.1088/0264-9381/9/10/014>, doi:10.1088/0264-9381/9/10/014
- [87] J. Friedman, M. S. Morris, I. D. Novikov, F. Echeverria, G. Klinkhammer, K. S. Thorne, and U. Yurtsever, “Cauchy problem in spacetimes with closed timelike curves”, *Physical Review D*, 42(6):1915–1930, September 1990. <https://link.aps.org/doi/10.1103/PhysRevD.42.1915>, doi:10.1103/PhysRevD.42.1915
- [88] J. L. Friedman and M. S. Morris, “The Cauchy problem for the scalar wave equation is well defined on a class of spacetimes with closed timelike curves”, *Physical Review Letters*, 66(4):401–404, January 1991. <https://link.aps.org/doi/10.1103/PhysRevLett.66.401>, doi:10.1103/PhysRevLett.66.401
- [89] J. L. Friedman and M. S. Morris, “Existence and Uniqueness Theorems for Massless Fields on a Class of Spacetimes with Closed Timelike Curves”, *Communications in Mathematical Physics*, 186(3):495–529, July 1997. <https://doi.org/10.1007/s002200050118>, doi:10.1007/s002200050118
- [90] J. L. Friedman, “The Cauchy Problem on Spacetimes That Are Not Globally Hyperbolic”, *The Einstein Equations and the Large Scale Behavior of Gravitational Fields*, pages 331–346, Birkhäuser, Basel, 2004. https://link.springer.com/chapter/10.1007/978-3-0348-7953-8_9, doi:10.1007/978-3-0348-7953-8_9

- [91] T. Nishitani, *Hyperbolic Systems with Analytic Coefficients: Well-posedness of the Cauchy Problem*, volume 2097 of Lecture Notes in Mathematics, Springer International Publishing, Cham, 2014.
- [92] V. Barbu, *Differential Equations*, Springer Undergraduate Mathematics Series, Springer Berlin Heidelberg, New York, NY, 2016.
- [93] A. Bachelot, “Global properties of the wave equation on non-globally hyperbolic manifolds”, *Journal de Mathématiques Pures et Appliquées*, 81(1):35–65, January 2002. <http://www.sciencedirect.com/science/article/pii/S0021782401012296>, doi:10.1016/S0021-7824(01)01229-6
- [94] I. Y. Aref'eva, T. Ishiwatari, and I. V. Volovich, “Cauchy problem on non-globally hyperbolic space-times”, *Theoretical and Mathematical Physics*, 157(3):1646–1654, December 2008. <http://link.springer.com/article/10.1007/s11232-008-0137-1>, doi:10.1007/s11232-008-0137-1
- [95] I. V. Volovich, O. V. Groshev, N. A. Gusev, and È. A. Kuryanovich, “On solutions to the wave equation on a non-globally hyperbolic manifold”, *Proceedings of the Steklov Institute of Mathematics*, 265(1):262–275, July 2009. <https://doi.org/10.1134/S0081543809020242>, doi:10.1134/S0081543809020242
- [96] O. V. Groshev, “Existence and uniqueness of classical solutions of the Cauchy problem on nonglobally hyperbolic manifolds”, *Theoretical and Mathematical Physics*, 164(3):1202–1207, September 2010. <https://doi.org/10.1007/s11232-010-0101-8>, doi:10.1007/s11232-010-0101-8
- [97] A. Bachelot, “The Klein–Gordon equation in the Anti-de Sitter cosmology”, *Journal de Mathématiques Pures et Appliquées*, 96(6):527–554, December 2011. <http://www.sciencedirect.com/science/article/pii/S0021782411000894>, doi:10.1016/j.matpur.2011.07.004
- [98] D. M. A. Bullock, “Klein-Gordon solutions on non-globally hyperbolic standard static spacetimes”, *Reviews in Mathematical Physics*, 24(10):1250028, November 2012. <https://www.worldscientific.com/doi/abs/10.1142/S0129055X12500286>, doi:10.1142/S0129055X12500286
- [99] F. Echeverría, G. Klinkhammer, and K. S. Thorne, “Billiard balls in wormhole spacetimes with closed timelike curves: Classical theory”, *Physical Review D*, 44(4):1077–1099, August 1991. <https://link.aps.org/doi/10.1103/PhysRevD.44.1077>, doi:10.1103/PhysRevD.44.1077
- [100] I. D. Novikov, “Time machine and self-consistent evolution in problems with self-interaction”, *Physical Review D*, 45(6):1989–1994, March 1992. <https://link.aps.org/doi/10.1103/PhysRevD.45.1989>, doi:10.1103/PhysRevD.45.1989
- [101] E. V. Mikheeva and I. D. Novikov, “Inelastic billiard ball in a spacetime with a time machine”, *Physical Review D*, 47(4):1432–1436, February 1993. <https://link.aps.org/doi/10.1103/PhysRevD.47.1432>, doi:10.1103/PhysRevD.47.1432
- [102] M. B. Mensky and I. D. Novikov, “Three-dimensional billiards with time machine”, *International Journal of Modern Physics D*, 05(02):179–192, April 1996. <https://www.worldscientific.com/doi/10.1142/S0218271896000126>, doi:10.1142/S0218271896000126
- [103] J. Dolanský and P. Krtouš, “Billiard ball in the space with a time machine”, *Physical Review D*, 82(12):124056, December 2010. <https://link.aps.org/doi/10.1103/PhysRevD.82.124056>, doi:10.1103/PhysRevD.82.124056
- [104] H. D. Politzer, “Simple quantum systems in spacetimes with closed timelike curves”, *Physical Review D*, 46(10):4470–4476, November 1992. <https://link.aps.org/doi/10.1103/PhysRevD.46.4470>, doi:10.1103/PhysRevD.46.4470
- [105] D. G. Boulware, “Quantum field theory in spaces with closed timelike curves”, *Physical Review D*, 46(10):4421–4441, November 1992. <https://link.aps.org/doi/10.1103/PhysRevD.46.4421>, doi:10.1103/PhysRevD.46.4421
- [106] H. D. Politzer, “Path integrals, density matrices, and information flow with closed timelike curves”, *Physical Review D*, 49(8):3981–3989, April 1994. <https://link.aps.org/doi/10.1103/PhysRevD.49.3981>, doi:10.1103/PhysRevD.49.3981
- [107] J. B. Hartle, “Unitarity and causality in generalized quantum mechanics for nonchronal spacetimes”, *Physical Review D*, 49(12):6543–6555, June 1994. <https://link.aps.org/doi/10.1103/PhysRevD.49.6543>, doi:10.1103/PhysRevD.49.6543

- [108] A. Anderson, “Unitarity restoration in the presence of closed timelike curves”, *Physical Review D*, 51(10):5707–5715, May 1995. <https://link.aps.org/doi/10.1103/PhysRevD.51.5707>, doi:10.1103/PhysRevD.51.5707
- [109] B. S. Kay, “The principle of locality and quantum field theory on (non globally hyperbolic) curved spacetimes”, *Reviews in Mathematical Physics*, 04(spec01):167–195, December 1992. <http://www.worldscientific.com/doi/abs/10.1142/S0129055X92000194>, doi:10.1142/S0129055X92000194
- [110] U. Yurtsever, “Algebraic approach to quantum field theory on non-globally-hyperbolic spacetimes”, *Classical and Quantum Gravity*, 11(4):999, April 1994. <https://iopscience.iop.org/article/10.1088/0264-9381/11/4/016/meta>, doi:10.1088/0264-9381/11/4/016
- [111] B. S. Kay, M. J. Radzikowski, and R. M. Wald, “Quantum Field Theory on Spacetimes with a Compactly Generated Cauchy Horizon”, *Communications in Mathematical Physics*, 183(3):533–556, February 1997. <https://doi.org/10.1007/s002200050042>, doi:10.1007/s002200050042
- [112] S. Krasnikov, “Quantum field theory and time machines”, *Physical Review D*, 59(2):024010, December 1998. <https://link.aps.org/doi/10.1103/PhysRevD.59.024010>, doi:10.1103/PhysRevD.59.024010
- [113] S. V. Sushkov, “Completeness principle and quantum field theory on nonglobally hyperbolic spacetimes”, *arXiv:gr-qc/9903046*, March 1999. <http://arxiv.org/abs/gr-qc/9903046>
- [114] I. Seggev, “Dynamics in stationary, non-globally hyperbolic spacetimes”, *Classical and Quantum Gravity*, 21(11):2651–2668, April 2004. <https://doi.org/10.1088%2F0264-9381%2F21%2F11%2F010>, doi:10.1088/0264-9381/21/11/010
- [115] C. J. Fewster, A. Higuchi, and C. G. Wells, “Classical and quantum initial value problems for models of chronology violation”, *Physical Review D*, 54(6):3806–3825, September 1996. <https://link.aps.org/doi/10.1103/PhysRevD.54.3806>, doi:10.1103/PhysRevD.54.3806
- [116] S. Lloyd, L. Maccone, R. Garcia-Patron, V. Giovannetti, Y. Shikano, S. Pirandola, L. A. Rozema, A. Darabi, Y. Soudagar, L. K. Shalm, and A. M. Steinberg, “Closed Timelike Curves via Postselection: Theory and Experimental Test of Consistency”, *Physical Review Letters*, 106(4):040403, January 2011. <https://link.aps.org/doi/10.1103/PhysRevLett.106.040403>, doi:10.1103/PhysRevLett.106.040403
- [117] M. Ringbauer, M. A. Broome, C. R. Myers, A. G. White, and T. C. Ralph, “Experimental simulation of closed timelike curves”, *Nature Communications*, 5(1):4145, September 2014. <http://www.nature.com/articles/ncomms5145>, doi:10.1038/ncomms5145
- [118] Y.-T. Huang, S.-W. Huang, J.-D. Lin, A. Miranowicz, N. Lambert, G.-Y. Chen, F. Nori, and Y.-N. Chen, “Experimental Decoding Scrambled Quantum Information from the Future”, January 2025. <http://arxiv.org/abs/2501.16335>, doi:10.48550/arXiv.2501.16335
- [119] J. L. Friedman, N. J. Papastamatiou, and J. Z. Simon, “Failure of unitarity for interacting fields on spacetimes with closed timelike curves”, *Physical Review D*, 46(10):4456–4469, November 1992. <https://link.aps.org/doi/10.1103/PhysRevD.46.4456>, doi:10.1103/PhysRevD.46.4456
- [120] D. S. Goldwirth, M. J. Perry, T. Piran, and K. S. Thorne, “Quantum propagator for a nonrelativistic particle in the vicinity of a time machine”, *Physical Review D*, 49(8):3951–3957, April 1994. <https://link.aps.org/doi/10.1103/PhysRevD.49.3951>, doi:10.1103/PhysRevD.49.3951
- [121] S. W. Hawking, “Quantum coherence and closed timelike curves”, *Physical Review D*, 52(10):5681–5686, November 1995. <https://link.aps.org/doi/10.1103/PhysRevD.52.5681>, doi:10.1103/PhysRevD.52.5681
- [122] C. J. Fewster and C. G. Wells, “Unitarity of quantum theory and closed timelike curves”, *Physical Review D*, 52(10):5773–5782, November 1995. <https://link.aps.org/doi/10.1103/PhysRevD.52.5773>, doi:10.1103/PhysRevD.52.5773
- [123] M. J. Cassidy, “Nonlinearity in quantum theory and closed timelike curves”, *Physical Review D*, 52(10):5676–5680, November 1995. <https://link.aps.org/doi/10.1103/PhysRevD.52.5676>, doi:10.1103/PhysRevD.52.5676

- [124] L. Gavassino, “Life on a closed timelike curve”, *Classical and Quantum Gravity*, 42(1):015002, December 2024. <https://dx.doi.org/10.1088/1361-6382/ad98df>, doi:10.1088/1361-6382/ad98df
- [125] A. Damle and T. Law, “A speculative model for cyclic information preservation in Kerr-Newman spacetime using closed timelike curves”, August 2024. <http://arxiv.org/abs/2408.02116>, doi:10.48550/arXiv.2408.02116
- [126] Y. Soen and A. Ori, “Improved time-machine model”, *Physical Review D*, 54(8):4858–4861, October 1996. <https://link.aps.org/doi/10.1103/PhysRevD.54.4858>, doi:10.1103/PhysRevD.54.4858
- [127] K. Lanczos, “On a Stationary Cosmology in the Sense of Einstein's Theory of Gravitation”, *General Relativity and Gravitation*, 29(3):363–399, March 1997. <https://link.springer.com/article/10.1023/A:1010277120072>, doi:10.1023/A:1010277120072
- [128] A. E. Everett and T. A. Roman, “Superluminal subway: The Krasnikov tube”, *Physical Review D*, 56(4):2100–2108, August 1997. <https://link.aps.org/doi/10.1103/PhysRevD.56.2100>, doi:10.1103/PhysRevD.56.2100
- [129] P. F. González-Díaz, “Kinks, energy conditions and closed timelike curves”, *International Journal of Modern Physics D*, 09(05):531–541, October 2000. <http://www.worldscientific.com/doi/abs/10.1142/S0218271800000475>, doi:10.1142/S0218271800000475
- [130] B. R. Steadman, “Causality Violation on van Stockum Geodesics”, *General Relativity and Gravitation*, 35(9):1721–1726, September 2003. <https://doi.org/10.1023/A:1025747605684>, doi:10.1023/A:1025747605684
- [131] V. E. Hubeny, M. Rangamani, and S. F. Ross, “Causal inheritance in plane wave quotients”, *Physical Review D*, 69(2):024007, January 2004. <https://link.aps.org/doi/10.1103/PhysRevD.69.024007>, doi:10.1103/PhysRevD.69.024007
- [132] P. Collas and D. Klein, “Causality Violating Geodesics in Bonnor's Rotating Dust Metric”, *General Relativity and Gravitation*, 36(11):2549–2557, November 2004. <https://doi.org/10.1023/B:GERG.0000046853.99745.e4>, doi:10.1023/B:GERG.0000046853.99745.e4
- [133] V. M. Rosa and P. S. Letelier, “Stability of closed timelike geodesics”, *Physics Letters A*, 370(2):99–103, October 2007. <https://www.sciencedirect.com/science/article/pii/S0375960107007578>, doi:10.1016/j.physleta.2007.05.041
- [134] Y. Duan, F. Liu, Y. Wang, and Y. C. Ong, “On the Counter-Rotation of Closed Timelike Curves”, *Universe*, 8(1):28, January 2022. <https://www.mdpi.com/2218-1997/8/1/28>, doi:10.3390/universe8010028
- [135] S. W. Hawking, “Chronology protection conjecture”, *Physical Review D*, 46(2):603–611, July 1992. <https://link.aps.org/doi/10.1103/PhysRevD.46.603>, doi:10.1103/PhysRevD.46.603
- [136] S. V. Sushkov, “Chronology protection and quantized fields: complex automorphic scalar field in Misner space”, *Classical and Quantum Gravity*, 14(2):523–533, February 1997. [https://doi.org/10.1088/0264-9381/14/2/025](https://doi.org/10.1088%2F0264-9381%2F14%2F2%2F025), doi:10.1088/0264-9381/14/2/025
- [137] J. L. Friedman, “Topological Censorship and Chronology Protection”, S. Dhurandhar and T. Padmanabhan (editors), *Gravitation and Cosmology*, Astrophysics and Space Science Library, pages 157–167, Dordrecht, 1997, Springer Netherlands. doi:10.1007/978-94-011-5812-1_11
- [138] L.-X. Li and J. R. Gott, “Self-Consistent Vacuum for Misner Space and the Chronology Protection Conjecture”, *Physical Review Letters*, 80(14):2980–2983, April 1998. <https://link.aps.org/doi/10.1103/PhysRevLett.80.2980>, doi:10.1103/PhysRevLett.80.2980
- [139] J. L. Friedman and A. Higuchi, “Topological censorship and chronology protection”, *arXiv:0801.0735 [gr-qc]*, 2008. <https://arxiv.org/abs/0801.0735>
- [140] G. Martín-Vázquez and C. Sabín, “Closed timelike curves and chronology protection in quantum and classical simulators”, *Classical and Quantum Gravity*, 37(4):045013, January 2020. <https://dx.doi.org/10.1088/1361-6382/ab5f3f>, doi:10.1088/1361-6382/ab5f3f
- [141] C. J. Fewster and A. Higuchi, “Quantum field theory on certain non-globally hyperbolic space-times”, *Classical and Quantum Gravity*, 13(1):51–61, January 1996. [https://doi.org/10.1088/0264-9381/13/1/006](https://doi.org/10.1088%2F0264-9381%2F13%2F1%2F006), doi:10.1088/0264-9381/13/1/006

- [142] M. Bartkiewicz, A. Grudka, R. Horodecki, J. Łodyga, and J. Wychowaniec, “Closed timelike curves and the second law of thermodynamics”, *Physical Review A*, 99(2):022304, February 2019. <https://link.aps.org/doi/10.1103/PhysRevA.99.022304>, doi:10.1103/PhysRevA.99.022304
- [143] C. Rovelli, “Can we travel to the past? Irreversible physics along closed timelike curves”, December 2019. <http://arxiv.org/abs/1912.04702>, doi:10.48550/arXiv.1912.04702
- [144] A. Ori and Y. Soen, “Causality violation and the weak energy condition”, *Physical Review D*, 49(8):3990–3997, April 1994. <https://link.aps.org/doi/10.1103/PhysRevD.49.3990>, doi:10.1103/PhysRevD.49.3990
- [145] F. Lobo and P. Crawford, “Time, Closed Timelike Curves and Causality”, R. Buccheri, M. Saniga, and W. M. Stuckey (editors), *The Nature of Time: Geometry, Physics and Perception*, NATO Science Series, pages 289–296, Springer Netherlands, Dordrecht, 2003. https://doi.org/10.1007/978-94-010-0155-7_30, doi:10.1007/978-94-010-0155-7_30
- [146] L. Hardy, “Probability Theories with Dynamic Causal Structure: A New Framework for Quantum Gravity”, *arXiv:gr-qc/0509120*, September 2005. <http://arxiv.org/abs/gr-qc/0509120>
- [147] E. Minguzzi and M. Sanchez, “The causal hierarchy of spacetimes”, *arXiv:gr-qc/0609119*, September 2006. <http://arxiv.org/abs/gr-qc/0609119>
- [148] L. Hardy, “Towards quantum gravity: a framework for probabilistic theories with non-fixed causal structure”, *Journal of Physics A: Mathematical and Theoretical*, 40(12):3081–3099, March 2007. [https://doi.org/10.1088/1751-8113/40/12/S12](https://doi.org/10.1088/1751-8113/40/12/s12), doi:10.1088/1751-8113/40/12/S12
- [149] J. Pearl, *Causality: Models, Reasoning, and Inference*, Cambridge University Press, Cambridge, 2nd edition, 2009.
- [150] F. S. N. Lobo, “Closed timelike curves and causality violation”, August 2010. <http://arxiv.org/abs/1008.1127>, doi:10.48550/arXiv.1008.1127
- [151] H. Price, “Does time-symmetry imply retrocausality? How the quantum world says ‘Maybe’?”, *Studies in History and Philosophy of Science Part B: Studies in History and Philosophy of Modern Physics*, 43(2):75–83, May 2012. <https://www.sciencedirect.com/science/article/pii/S1355219811000736>, doi:10.1016/j.shpsb.2011.12.003
- [152] O. Oreshkov, F. Costa, and Č. Brukner, “Quantum correlations with no causal order”, *Nature Communications*, 3(1):1092, October 2012. <http://www.nature.com/articles/ncomms2076>, doi:10.1038/ncomms2076
- [153] G. Chiribella, G. M. D’Ariano, P. Perinotti, and B. Valiron, “Quantum computations without definite causal structure”, *Physical Review A*, 88(2):022318, August 2013. <https://link.aps.org/doi/10.1103/PhysRevA.88.022318>, doi:10.1103/PhysRevA.88.022318
- [154] Ä. Baumeler, A. Feix, and S. Wolf, “Maximal incompatibility of locally classical behavior and global causal order in multiparty scenarios”, *Physical Review A*, 90(4):042106, October 2014. <https://link.aps.org/doi/10.1103/PhysRevA.90.042106>, doi:10.1103/PhysRevA.90.042106
- [155] M. Araújo, C. Branciard, F. Costa, A. Feix, C. Giarmatzi, and Č. Brukner, “Witnessing causal nonseparability”, *New Journal of Physics*, 17(10):102001, October 2015. <https://doi.org/10.1088/1367-2630/17/10/102001>, doi:10.1088/1367-2630/17/10/102001
- [156] C. Branciard, M. Araújo, A. Feix, F. Costa, and Č. Brukner, “The simplest causal inequalities and their violation”, *New Journal of Physics*, 18(1):013008, December 2015. <https://doi.org/10.1088/1367-2630/18/1/013008>, doi:10.1088/1367-2630/18/1/013008
- [157] X. Yuan, S. M. Assad, J. Thompson, J. Y. Haw, V. Vedral, T. C. Ralph, P. K. Lam, C. Weedbrook, and M. Gu, “Replicating the benefits of Deutschian closed timelike curves without breaking causality”, *npj Quantum Information*, 1(1):15007, December 2015. <http://www.nature.com/articles/npjqi20157>, doi:10.1038/npjqi.2015.7
- [158] Ä. Baumeler and S. Wolf, “The space of logically consistent classical processes without causal order”, *New Journal of Physics*, 18(1):013036, January 2016. <https://doi.org/10.1088/1367-2630/18/1/013036>, doi:10.1088/1367-2630/18/1/013036

- [159] Y. Aharonov, E. Cohen, and T. Shushi, “Accommodating Retrocausality with Free Will”, *Quanta*, 5(1):53–60, January 2016. <http://quanta.ws/ojs/index.php/quanta/article/view/44>, doi:10.12743/quanta.v5i1.44
- [160] Ä. Baumeler and S. Wolf, “Device-independent test of causal order and relations to fixed-points”, *New Journal of Physics*, 18(3):035014, April 2016. <https://doi.org/10.1088/1367-2630/18/3/035014>, doi:10.1088/1367-2630/18/3/035014
- [161] F. Costa and S. Shrapnel, “Quantum causal modelling”, *New Journal of Physics*, 18(6):063032, June 2016. <https://doi.org/10.1088/1367-2630/18/6/063032>, doi:10.1088/1367-2630/18/6/063032
- [162] V. Baumann and Č. Brukner, “Appearance of causality in process matrices when performing fixed-basis measurements for two parties”, *Physical Review A*, 93(6):062324, June 2016. <https://link.aps.org/doi/10.1103/PhysRevA.93.062324>, doi:10.1103/PhysRevA.93.062324
- [163] O. Oreshkov and C. Giarmatzi, “Causal and causally separable processes”, *New Journal of Physics*, 18(9):093020, September 2016. <https://doi.org/10.1088/1367-2630/18/9/093020>, doi:10.1088/1367-2630/18/9/093020
- [164] A. A. Abbott, C. Giarmatzi, F. Costa, and C. Branciard, “Multipartite causal correlations: Polytopes and inequalities”, *Physical Review A*, 94(3):032131, September 2016. <https://link.aps.org/doi/10.1103/PhysRevA.94.032131>, doi:10.1103/PhysRevA.94.032131
- [165] F. Giacomini, E. Castro-Ruiz, and Č. Brukner, “Indefinite causal structures for continuous-variable systems”, *New Journal of Physics*, 18(11):113026, November 2016. <https://doi.org/10.1088/1367-2630/18/11/113026>, doi:10.1088/1367-2630/18/11/113026
- [166] P. Perinotti, “Causal Structures and the Classification of Higher Order Quantum Computations”, R. Renner and S. Stupar (editors), *Time in Physics*, Tutorials, Schools, and Workshops in the Mathematical Sciences, pages 103–127, Springer International Publishing, Cham, 2017. https://link.springer.com/chapter/10.1007/978-3-319-68655-4_7, doi:10.1007/978-3-319-68655-4_7
- [167] M. Araújo, A. Feix, M. Navascués, and Č. Brukner, “A purification postulate for quantum mechanics with indefinite causal order”, *Quantum*, 1:10, April 2017. <https://quantum-journal.org/papers/q-2017-04-26-10/>, doi:10.22331/q-2017-04-26-10
- [168] R. I. Sutherland, “How retrocausality helps”, *AIP Conference Proceedings*, 1841(1):020001, May 2017. <https://aip.scitation.org/doi/abs/10.1063/1.4982765>, doi:10.1063/1.4982765
- [169] M. Araújo, P. A. Guérin, and Ä. Baumeler, “Quantum computation with indefinite causal structures”, *Physical Review A*, 96(5):052315, November 2017. <https://link.aps.org/doi/10.1103/PhysRevA.96.052315>, doi:10.1103/PhysRevA.96.052315
- [170] A. A. Abbott, J. Wechs, F. Costa, and C. Branciard, “Genuinely multipartite noncausality”, *Quantum*, 1:39, December 2017. <https://quantum-journal.org/papers/q-2017-12-14-39/>, doi:10.22331/q-2017-12-14-39
- [171] A. Kumar, I. Chakrabarty, A. K. Pati, A. Sen(De), and U. Sen, “Quantum no-go theorems in causality respecting systems in the presence of closed timelike curves: Tweaking the Deutsch condition”, *EPL (Europhysics Letters)*, 122(1):10007, June 2018. [https://doi.org/10.1209/0295-5075/122/10007](https://doi.org/10.1209%2F0295-5075%2F122%2F10007), doi:10.1209/0295-5075/122/10007
- [172] K. Wharton, “A New Class of Retrocausal Models”, *Entropy*, 20(6):410, June 2018. <https://www.mdpi.com/1099-4300/20/6/410>, doi:10.3390/e20060410
- [173] D. Jia, “Quantum theories from principles without assuming a definite causal structure”, *Physical Review A*, 98(3):032112, September 2018. <https://link.aps.org/doi/10.1103/PhysRevA.98.032112>, doi:10.1103/PhysRevA.98.032112
- [174] E. Minguzzi, “Lorentzian causality theory”, *Living Reviews in Relativity*, 22(1):3, June 2019. <https://doi.org/10.1007/s41114-019-0019-x>, doi:10.1007/s41114-019-0019-x
- [175] D. Jia and F. Costa, “Causal order as a resource for quantum communication”, *Physical Review A*, 100(5):052319, November 2019. <https://link.aps.org/doi/10.1103/PhysRevA.100.052319>, doi:10.1103/PhysRevA.100.052319

- [176] K. D. Olum, “Superluminal Travel Requires Negative Energies”, *Physical Review Letters*, 81(17):3567–3570, October 1998. <https://link.aps.org/doi/10.1103/PhysRevLett.81.3567>, doi:10.1103/PhysRevLett.81.3567
- [177] F. Lobo and P. Crawford, “Weak energy condition violation and superluminal travel”, *Current Trends in Relativistic Astrophysics*, volume 617 of Lecture Notes in Physics, pages 277–291, 2003. <http://arxiv.org/abs/gr-qc/0204038>, doi:10.1007/3-540-36973-2_15
- [178] J. J. Wallman and S. D. Bartlett, “Revisiting Consistency Conditions for Quantum States of Systems on Closed Timelike Curves: An Epistemic Perspective”, *Foundations of Physics*, 42(5):656–673, May 2012. <https://doi.org/10.1007/s10701-012-9635-y>, doi:10.1007/s10701-012-9635-y
- [179] C. Mallary, G. Khanna, and R. H. Price, “Closed timelike curves and ‘effective’ superluminal travel with naked line singularities”, *Classical and Quantum Gravity*, 35(17):175020, August 2018. <https://doi.org/10.1088/1361-6382/aad306>, doi:10.1088/1361-6382/aad306
- [180] S. Gutti, S. Kulkarni, and V. Prasad, “Closed timelike curves and energy conditions in regular spacetimes”, *The European Physical Journal C*, 82(12):1136, December 2022. <https://doi.org/10.1140/epjc/s10052-022-11114-1>, doi:10.1140/epjc/s10052-022-11114-1
- [181] F. S. N. Lobo, “Exotic solutions in General Relativity: Traversable wormholes and ‘warp drive’ spacetimes”, October 2007. <http://arxiv.org/abs/0710.4474>, doi:10.48550/arXiv.0710.4474
- [182] J. Bub and A. Stairs, “Quantum interactions with closed timelike curves and superluminal signaling”, *Physical Review A*, 89(2):022311, February 2014. <https://link.aps.org/doi/10.1103/PhysRevA.89.022311>, doi:10.1103/PhysRevA.89.022311
- [183] R. Emparan and M. Tomašević, “Holography of time machines”, *Journal of High Energy Physics*, 2022(3):212, March 2022. [https://doi.org/10.1007/JHEP03\(2022\)212](https://doi.org/10.1007/JHEP03(2022)212), doi:10.1007/JHEP03(2022)212
- [184] D. Deutsch, “Quantum mechanics near closed timelike lines”, *Physical Review D*, 44(10):3197–3217, November 1991. <https://link.aps.org/doi/10.1103/PhysRevD.44.3197>, doi:10.1103/PhysRevD.44.3197
- [185] T. C. Ralph and C. R. Myers, “Information flow of quantum states interacting with closed timelike curves”, *Physical Review A*, 82(6):062330, December 2010. <https://link.aps.org/doi/10.1103/PhysRevA.82.062330>, doi:10.1103/PhysRevA.82.062330
- [186] T. C. Ralph and C. R. Myers, “Reply to ‘Comment on ‘Information flow of quantum states interacting with closed timelike curves’””, *Physical Review A*, 84(5):056302, November 2011. <https://link.aps.org/doi/10.1103/PhysRevA.84.056302>, doi:10.1103/PhysRevA.84.056302
- [187] J. L. Pienaar, C. R. Myers, and T. C. Ralph, “Quantum fields on closed timelike curves”, *Physical Review A*, 84(6):062316, December 2011. <https://link.aps.org/doi/10.1103/PhysRevA.84.062316>, doi:10.1103/PhysRevA.84.062316
- [188] X. Dong, H. Chen, and L. Zhou, “Ralph’s equivalent circuit model, revised Deutsch’s maximum entropy rule and discontinuous quantum evolutions in D-CTCs”, *arXiv:1711.06814 [quant-ph]*, November 2017. <http://arxiv.org/abs/1711.06814>
- [189] S. Lloyd, L. Maccone, R. Garcia-Patron, V. Giovannetti, and Y. Shikano, “Quantum mechanics of time travel through post-selected teleportation”, *Physical Review D*, 84(2):025007, July 2011. <https://link.aps.org/doi/10.1103/PhysRevD.84.025007>, doi:10.1103/PhysRevD.84.025007
- [190] C. H. Bennett and B. Schumacher, “Simulated time travel, teleportation without communication, and how to conduct a romance with someone who has fallen into a black hole”, May 2005. <http://www.research.ibm.com/people/b/bennetc/QUPONBshort.pdf>
- [191] G. Svetlichny, “Effective Quantum Time Travel”, 50(12):3903–3914, February 2009. <http://arxiv.org/abs/0902.4898>
- [192] G. Svetlichny, “Time Travel: Deutsch vs. Teleportation”, *International Journal of Theoretical Physics*, 50(12):3903–3914, December 2011. <https://doi.org/10.1007/s10773-011-0973-x>, doi:10.1007/s10773-011-0973-x

- [193] D. F. V. James, P. G. Kwiat, W. J. Munro, and A. G. White, “Measurement of qubits”, *Physical Review A*, 64(5):052312, October 2001. <https://link.aps.org/doi/10.1103/PhysRevA.64.052312>, doi:10.1103/PhysRevA.64.052312
- [194] G. Mauro D’Ariano, M. G. A. Paris, and M. F. Sacchi, “Quantum Tomography”, P. W. Hawkes (editor), *Advances in Imaging and Electron Physics*, volume 128, pages 205–308, Elsevier, January 2003. <https://www.sciencedirect.com/science/article/pii/S1076567003800654>, doi:10.1016/S1076-5670(03)80065-4
- [195] G. M. D’Ariano, M. G.A. Paris, and M. F. Sacchi, “2 Quantum Tomographic Methods”, M. Paris and J. Řeháček (editors), *Quantum State Estimation*, Lecture Notes in Physics, pages 7–58, Springer, Berlin, Heidelberg, 2004. https://doi.org/10.1007/978-3-540-44481-7_2, doi:10.1007/978-3-540-44481-7_2
- [196] Y. Aharonov and D. Rohrlich, *Quantum Paradoxes: Quantum Theory for the Perplexed*, Wiley, 1st edition, February 2005.
- [197] Y. Aharonov, D. Z. Albert, and L. Vaidman, “How the result of a measurement of a component of the spin of a spin-1/2 particle can turn out to be 100”, *Physical Review Letters*, 60(14):1351–1354, April 1988. <https://link.aps.org/doi/10.1103/PhysRevLett.60.1351>, doi:10.1103/PhysRevLett.60.1351
- [198] A. Peres, “Quantum measurements with postselection”, *Physical Review Letters*, 62(19):2326–2326, May 1989. <https://link.aps.org/doi/10.1103/PhysRevLett.62.2326>, doi:10.1103/PhysRevLett.62.2326
- [199] A. J. Leggett, “Comment on “How the result of a measurement of a component of the spin of a spin-1/2 particle can turn out to be 100””, *Physical Review Letters*, 62(19):2325–2325, May 1989. <https://link.aps.org/doi/10.1103/PhysRevLett.62.2325>, doi:10.1103/PhysRevLett.62.2325
- [200] Y. Aharonov and L. Vaidman, “Aharonov and Vaidman reply”, *Physical Review Letters*, 62(19):2327–2327, May 1989. <https://link.aps.org/doi/10.1103/PhysRevLett.62.2327>, doi:10.1103/PhysRevLett.62.2327
- [201] I. M. Duck, P. M. Stevenson, and E. C. G. Sudarshan, “The sense in which a “weak measurement” of a spin-1/2 particle’s spin component yields a value 100”, *Physical Review D*, 40(6):2112–2117, September 1989. <https://link.aps.org/doi/10.1103/PhysRevD.40.2112>, doi:10.1103/PhysRevD.40.2112
- [202] Y. Aharonov and L. Vaidman, “Properties of a quantum system during the time interval between two measurements”, *Physical Review A*, 41(1):11–20, January 1990. <https://link.aps.org/doi/10.1103/PhysRevA.41.11>, doi:10.1103/PhysRevA.41.11
- [203] J. M. Knight and L. Vaidman, “Weak measurement of photon polarization”, *Physics Letters A*, 143(8):357–361, January 1990. <https://www.sciencedirect.com/science/article/pii/037596019090371T>, doi:10.1016/0375-9601(90)90371-T
- [204] J.g. Story, N.w.m. Ritchie, and R.g. Hulet, “Weak measurements”, *Modern Physics Letters B*, 05(26):1713–1725, November 1991. <https://www.worldscientific.com/doi/abs/10.1142/S0217984991002069>, doi:10.1142/S0217984991002069
- [205] L. M. Johansen, “Weak Measurements with Arbitrary Probe States”, *Physical Review Letters*, 93(12):120402, September 2004. <https://link.aps.org/doi/10.1103/PhysRevLett.93.120402>, doi:10.1103/PhysRevLett.93.120402
- [206] L. Vaidman, “Weak Value and Weak Measurements”, D. Greenberger, K. Hentschel, and F. Weinert (editors), *Compendium of Quantum Physics*, pages 840–842, Springer, Berlin, Heidelberg, 2009. https://doi.org/10.1007/978-3-540-70626-7_235, doi:10.1007/978-3-540-70626-7_235
- [207] S. Wu and K. Mølmer, “Weak measurements with a qubit meter”, *Physics Letters A*, 374(1):34–39, December 2009. <https://www.sciencedirect.com/science/article/pii/S0375960109012882>, doi:10.1016/j.physleta.2009.10.026
- [208] Y. Kedem and L. Vaidman, “Modular Values and Weak Values of Quantum Observables”, *Physical Review Letters*, 105(23):230401, November 2010. <https://link.aps.org/doi/10.1103/PhysRevLett.105.230401>, doi:10.1103/PhysRevLett.105.230401

- [209] S. Wu and Y. Li, “Weak measurements beyond the Aharonov-Albert-Vaidman formalism”, *Physical Review A*, 83(5):052106, May 2011. <https://link.aps.org/doi/10.1103/PhysRevA.83.052106>, doi:10.1103/PhysRevA.83.052106
- [210] K. Nakamura, A. Nishizawa, and M.-K. Fujimoto, “Evaluation of weak measurements to all orders”, *Physical Review A*, 85(1):012113, January 2012. <https://link.aps.org/doi/10.1103/PhysRevA.85.012113>, doi:10.1103/PhysRevA.85.012113
- [211] J. S. Lundeen and C. Bamber, “Procedure for Direct Measurement of General Quantum States Using Weak Measurement”, *Physical Review Letters*, 108(7):070402, February 2012. <https://link.aps.org/doi/10.1103/PhysRevLett.108.070402>, doi:10.1103/PhysRevLett.108.070402
- [212] S. Pang, S. Wu, and Z.-B. Chen, “Weak measurement with orthogonal preselection and postselection”, *Physical Review A*, 86(2):022112, August 2012. <https://link.aps.org/doi/10.1103/PhysRevA.86.022112>, doi:10.1103/PhysRevA.86.022112
- [213] A. G. Kofman, S. Ashhab, and F. Nori, “Nonperturbative theory of weak pre- and post-selected measurements”, *Physics Reports*, 520(2):43–133, November 2012. <https://www.sciencedirect.com/science/article/pii/S0370157312002050>, doi:10.1016/j.physrep.2012.07.001
- [214] L. G. Bishop, F. Costa, and T. C. Ralph, “Quantum state tomography on closed timelike curves using weak measurements”, *Classical and Quantum Gravity*, 42(4):045018, February 2025. <https://dx.doi.org/10.1088/1361-6382/ada90b>, doi:10.1088/1361-6382/ada90b
- [215] A. K. Pati, I. Chakrabarty, and P. Agrawal, “Purification of mixed states with closed timelike curve is not possible”, *Physical Review A*, 84(6):062325, December 2011. <https://link.aps.org/doi/10.1103/PhysRevA.84.062325>, doi:10.1103/PhysRevA.84.062325
- [216] S. Ghosh, A. Adhikary, and G. Paul, “Quantum signaling to the past using P-CTCS”, *Quantum Information and Computation*, 18(11&12):965–974, September 2018. <http://www.rintonpress.com/journals/doi/QIC18.11-12-5.html>, doi:10.26421/QIC18.11-12-5
- [217] L. G. Bishop, F. Costa, and T. C. Ralph, “Time-traveling billiard-ball clocks: A quantum model”, *Physical Review A*, 103(4):042223, April 2021. <https://link.aps.org/doi/10.1103/PhysRevA.103.042223>, doi:10.1103/PhysRevA.103.042223
- [218] T. A. Brun, “Computers with Closed Timelike Curves Can Solve Hard Problems Efficiently”, *Foundations of Physics Letters*, 16(3):245–253, June 2003. <https://doi.org/10.1023/A:1025967225931>, doi:10.1023/A:1025967225931
- [219] S. Aaronson and J. Watrous, “Closed timelike curves make quantum and classical computing equivalent”, *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 465(2102):631–647, February 2009. <https://royalsocietypublishing.org/doi/10.1098/rspa.2008.0350>, doi:10.1098/rspa.2008.0350
- [220] D. Bacon, “Quantum computational complexity in the presence of closed timelike curves”, *Physical Review A*, 70(3):032309, September 2004. <https://link.aps.org/doi/10.1103/PhysRevA.70.032309>, doi:10.1103/PhysRevA.70.032309
- [221] T. A. Brun and M. M. Wilde, “Perfect State Distinguishability and Computational Speedups with Postselected Closed Timelike Curves”, *Foundations of Physics*, 42(3):341–361, March 2012. <https://doi.org/10.1007/s10701-011-9601-0>, doi:10.1007/s10701-011-9601-0
- [222] S. Aaronson, “Quantum computing, postselection, and probabilistic polynomial-time”, *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 461(2063):3473–3482, November 2005. <https://royalsocietypublishing.org/doi/10.1098/rspa.2005.1546>, doi:10.1098/rspa.2005.1546
- [223] T. A. Brun, J. Harrington, and M. M. Wilde, “Localized Closed Timelike Curves Can Perfectly Distinguish Quantum States”, *Physical Review Letters*, 102(21):210402, May 2009. <https://link.aps.org/doi/10.1103/PhysRevLett.102.210402>, doi:10.1103/PhysRevLett.102.210402
- [224] D. Ahn, C. R. Myers, T. C. Ralph, and R. B. Mann, “Quantum-state cloning in the presence of a closed timelike curve”, *Physical Review A*, 88(2):022332, August 2013. <https://link.aps.org/doi/10.1103/PhysRevA.88.022332>, doi:10.1103/PhysRevA.88.022332

- [225] T. A. Brun, M. M. Wilde, and A. Winter, “Quantum State Cloning Using Deutschian Closed Timelike Curves”, *Physical Review Letters*, 111(19):190401, November 2013. <https://link.aps.org/doi/10.1103/PhysRevLett.111.190401>, doi:10.1103/PhysRevLett.111.190401
- [226] D. T. Pegg, “Quantum mechanics and the time travel paradox”, June 2005. <http://arxiv.org/abs/quant-ph/0506141>, doi:10.48550/arXiv.quant-ph/0506141
- [227] D. M. Greenberger and K. Svozil, “Quantum Theory Looks at Time Travel”, A. C. Elitzur, S. Dolev, and N. Kolenda (editors), *Quo Vadis Quantum Mechanics?*, The Frontiers Collection, pages 63–71, Springer, Berlin, Heidelberg, 2005. https://doi.org/10.1007/3-540-26669-0_4, doi:10.1007/3-540-26669-0_4
- [228] M. Czachor, “Time travel without paradoxes: Ring resonator as a universal paradigm for looped quantum evolutions”, *Physics Letters A*, 383(23):2704–2712, August 2019. <http://www.sciencedirect.com/science/article/pii/S0375960119304943>, doi:10.1016/j.physleta.2019.05.043
- [229] P. E. Gibbs, “The Small Scale Structure of Space-Time: A Bibliographical Review”, January 1996. <http://arxiv.org/abs/hep-th/9506171>, doi:10.48550/arXiv.hep-th/9506171
- [230] F. Piazza, “Glimmers of a Pre-geometric Perspective”, *Foundations of Physics*, 40(3):239–266, March 2010. <https://doi.org/10.1007/s10701-009-9387-5>, doi:10.1007/s10701-009-9387-5
- [231] M. Zych, F. Costa, I. Pikovski, and Č. Brukner, “Bell’s theorem for temporal order”, *Nature Communications*, 10(1):3772, August 2019. <http://www.nature.com/articles/s41467-019-11579-x>, doi:10.1038/s41467-019-11579-x
- [232] A. Feix, M. Araújo, and Č. Brukner, “Quantum superposition of the order of parties as a communication resource”, *Physical Review A*, 92(5):052326, November 2015. <https://link.aps.org/doi/10.1103/PhysRevA.92.052326>, doi:10.1103/PhysRevA.92.052326
- [233] S. Shrapnel, F. Costa, and G. Milburn, “Updating the Born rule”, *New Journal of Physics*, 20(5):053010, May 2018. <https://doi.org/10.1088/1367-2630/aabe12>, doi:10.1088/1367-2630/aabe12
- [234] J. Bavaresco, M. Araújo, Č. Brukner, and M. T. Quintino, “Semi-device-independent certification of indefinite causal order”, *Quantum*, 3:176, August 2019. <https://quantum-journal.org/papers/q-2019-08-19-176/>, doi:10.22331/q-2019-08-19-176
- [235] Ä. Baumeler, F. Costa, T. C. Ralph, S. Wolf, and M. Zych, “Reversible time travel with freedom of choice”, *Classical and Quantum Gravity*, 36(22):224002, October 2019. <https://doi.org/10.1088%2F1361-6382%2Fab4973>, doi:10.1088/1361-6382/ab4973
- [236] G. Tobar and F. Costa, “Reversible dynamics with closed time-like curves and freedom of choice”, *Classical and Quantum Gravity*, 37(20):205011, September 2020. <https://doi.org/10.1088%2F1361-6382%2Faba4bc>, doi:10.1088/1361-6382/aba4bc
- [237] V. Vedral, A. Barenco, and A. Ekert, “Quantum networks for elementary arithmetic operations”, *Physical Review A*, 54(1):147–153, July 1996. <https://link.aps.org/doi/10.1103/PhysRevA.54.147>, doi:10.1103/PhysRevA.54.147
- [238] T. G. Draper, “Addition on a Quantum Computer”, August 2000. <http://arxiv.org/abs/quant-ph/0008033>, doi:10.48550/arXiv.quant-ph/0008033
- [239] L. Ruiz-Perez and J. C. Garcia-Escartin, “Quantum arithmetic with the quantum Fourier transform”, *Quantum Information Processing*, 16(6):152, April 2017. <https://doi.org/10.1007/s11128-017-1603-1>, doi:10.1007/s11128-017-1603-1
- [240] E. Şahin, “Quantum arithmetic operations based on quantum fourier transform on signed integers”, *International Journal of Quantum Information*, 18(06):2050035, September 2020. <https://www.worldscientific.com/doi/abs/10.1142/S0219749920500355>, doi:10.1142/S0219749920500355
- [241] R. Seidel, N. Tcholtchev, S. Bock, C. K.-U. Becker, and M. Hauswirth, “Efficient Floating Point Arithmetic for Quantum Computers”, *IEEE Access*, 10:72400–72415, 2022. <https://ieeexplore.ieee.org/document/9815035>, doi:10.1109/ACCESS.2022.3188251
- [242] P. Atchade-Adelomou and S. Gonzalez, “Efficient Quantum Modular Arithmetics for the ISQ Era”, November 2023. <http://arxiv.org/abs/2311.08555>, doi:10.48550/arXiv.2311.08555

- [243] S. Wang, X. Li, W. J. B. Lee, S. Deb, E. Lim, and A. Chattopadhyay, “A comprehensive study of quantum arithmetic circuits”, *Philosophical Transactions A*, January 2025. <https://royalsocietypublishing.org/doi/10.1098/rsta.2023.0392>, doi:10.1098/rsta.2023.0392
- [244] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, “Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels”, *Physical Review Letters*, 70(13):1895–1899, 1993. <https://link.aps.org/doi/10.1103/PhysRevLett.70.1895>, doi:10.1103/PhysRevLett.70.1895
- [245] O. Hosten and P. Kwiat, “Observation of the Spin Hall Effect of Light via Weak Measurements”, *Science*, 319(5864):787–790, February 2008. <https://www.science.org/doi/10.1126/science.1152697>, doi:10.1126/science.1152697
- [246] J. S. Lundeen and A. M. Steinberg, “Experimental Joint Weak Measurement on a Photon Pair as a Probe of Hardy's Paradox”, *Physical Review Letters*, 102(2):020404, January 2009. <https://link.aps.org/doi/10.1103/PhysRevLett.102.020404>, doi:10.1103/PhysRevLett.102.020404
- [247] P. B. Dixon, D. J. Starling, A. N. Jordan, and J. C. Howell, “Ultrasensitive Beam Deflection Measurement via Interferometric Weak Value Amplification”, *Physical Review Letters*, 102(17):173601, April 2009. <https://link.aps.org/doi/10.1103/PhysRevLett.102.173601>, doi:10.1103/PhysRevLett.102.173601
- [248] Y.-S. Kim, Y.-W. Cho, Y.-S. Ra, and Y.-H. Kim, “Reversing the weak quantum measurement for a photonic qubit”, *Optics Express*, 17(14):11978–11985, July 2009. <https://opg.optica.org/oe/abstract.cfm?uri=oe-17-14-11978>, doi:10.1364/OE.17.011978
- [249] Y.-W. Cho, H.-T. Lim, Y.-S. Ra, and Y.-H. Kim, “Weak value measurement with an incoherent measuring device”, *New Journal of Physics*, 12(2):023036, February 2010. <https://dx.doi.org/10.1088/1367-2630/12/2/023036>, doi:10.1088/1367-2630/12/2/023036
- [250] M. E. Goggin, M. P. Almeida, M. Barbieri, B. P. Lanyon, J. L. O'Brien, A. G. White, and G. J. Pryde, “Violation of the Leggett–Garg inequality with weak measurements of photons”, *Proceedings of the National Academy of Sciences*, 108(4):1256–1261, January 2011. <https://www.pnas.org/doi/full/10.1073/pnas.1005774108>, doi:10.1073/pnas.1005774108
- [251] O. Zilberberg, A. Romito, and Y. Gefen, “Charge Sensing Amplification via Weak Values Measurement”, *Physical Review Letters*, 106(8):080405, February 2011. <https://link.aps.org/doi/10.1103/PhysRevLett.106.080405>, doi:10.1103/PhysRevLett.106.080405
- [252] J. S. Lundeen, B. Sutherland, A. Patel, C. Stewart, and C. Bamber, “Direct measurement of the quantum wavefunction”, *Nature*, 474(7350):188–191, June 2011. <http://www.nature.com/articles/nature10120>, doi:10.1038/nature10120
- [253] S. Kocsis, B. Braverman, S. Ravets, M. J. Stevens, R. P. Mirin, L. K. Shalm, and A. M. Steinberg, “Observing the Average Trajectories of Single Photons in a Two-Slit Interferometer”, *Science*, 332(6034):1170–1173, June 2011. <https://www.science.org/doi/10.1126/science.1202218>, doi:10.1126/science.1202218
- [254] A. Feizpour, X. Xing, and A. M. Steinberg, “Amplifying Single-Photon Nonlinearity Using Weak Measurements”, *Physical Review Letters*, 107(13):133603, September 2011. <https://link.aps.org/doi/10.1103/PhysRevLett.107.133603>, doi:10.1103/PhysRevLett.107.133603
- [255] Y.-S. Kim, J.-C. Lee, O. Kwon, and Y.-H. Kim, “Protecting entanglement from decoherence using weak measurement and quantum measurement reversal”, *Nature Physics*, 8(2):117–120, February 2012. <http://www.nature.com/articles/nphys2178>, doi:10.1038/nphys2178
- [256] L. A. Rozema, A. Darabi, D. H. Mahler, A. Hayat, Y. Soudagar, and A. M. Steinberg, “Violation of Heisenberg's Measurement-Disturbance Relationship by Weak Measurements”, *Physical Review Letters*, 109(10):100404, September 2012. <https://link.aps.org/doi/10.1103/PhysRevLett.109.100404>, doi:10.1103/PhysRevLett.109.100404
- [257] R. Vijay, C. Macklin, D. H. Slichter, S. J. Weber, K. W. Murch, R. Naik, A. N. Korotkov, and I. Siddiqi, “Stabilizing Rabi oscillations in a superconducting qubit using quantum feedback”, *Nature*, 490(7418):77–80, October 2012. <http://www.nature.com/articles/nature11505>, doi:10.1038/nature11505

- [258] M. Hatridge, S. Shankar, M. Mirrahimi, F. Schackert, K. Geerlings, T. Brecht, K. M. Sliwa, B. Abdo, L. Frunzio, S. M. Girvin, R. J. Schoelkopf, and M. H. Devoret, “Quantum Back-Action of an Individual Variable-Strength Measurement”, *Science*, 339(6116):178–181, January 2013. <https://www.science.org/doi/10.1126/science.1226897>, doi:10.1126/science.1226897
- [259] J. Z. Salvail, M. Agnew, A. S. Johnson, E. Bolduc, J. Leach, and R. W. Boyd, “Full characterization of polarization states of light via direct measurement”, *Nature Photonics*, 7(4):316–321, April 2013. <http://www.nature.com/articles/nphoton.2013.24>, doi:10.1038/nphoton.2013.24
- [260] J. P. Groen, D. Ristè, L. Tornberg, J. Cramer, P. C. de Groot, T. Picot, G. Johansson, and L. DiCarlo, “Partial-Measurement Backaction and Nonclassical Weak Values in a Superconducting Circuit”, *Physical Review Letters*, 111(9):090506, August 2013. <https://link.aps.org/doi/10.1103/PhysRevLett.111.090506>, doi:10.1103/PhysRevLett.111.090506
- [261] M. Malik, M. Mirhosseini, M. P. J. Lavery, J. Leach, M. J. Padgett, and R. W. Boyd, “Direct measurement of a 27-dimensional orbital-angular-momentum state vector”, *Nature Communications*, 5(1):3115, January 2014. <http://www.nature.com/articles/ncomms4115>, doi:10.1038/ncomms4115
- [262] M. S. Blok, C. Bonato, M. L. Markham, D. J. Twitchen, V. V. Dobrovitski, and R. Hanson, “Manipulating a qubit through the backaction of sequential partial measurements and real-time feedback”, *Nature Physics*, 10(3):189–193, March 2014. <http://www.nature.com/articles/nphys2881>, doi:10.1038/nphys2881
- [263] O. S. Magaña-Loaiza, M. Mirhosseini, B. Rodenburg, and R. W. Boyd, “Amplification of Angular Rotations Using Weak Measurements”, *Physical Review Letters*, 112(20):200401, May 2014. <https://link.aps.org/doi/10.1103/PhysRevLett.112.200401>, doi:10.1103/PhysRevLett.112.200401
- [264] T. Denkmayr, H. Geppert, S. Sponar, H. Lemmel, A. Matzkin, J. Tollaksen, and Y. Hasegawa, “Observation of a quantum Cheshire Cat in a matter-wave interferometer experiment”, *Nature Communications*, 5(1):4492, July 2014. <http://www.nature.com/articles/ncomms5492>, doi:10.1038/ncomms5492
- [265] M. Mirhosseini, O. S. Magaña-Loaiza, S. M. Hashemi Rafsanjani, and R. W. Boyd, “Compressive Direct Measurement of the Quantum Wave Function”, *Physical Review Letters*, 113(9):090402, August 2014. <https://link.aps.org/doi/10.1103/PhysRevLett.113.090402>, doi:10.1103/PhysRevLett.113.090402
- [266] Z. Shi, M. Mirhosseini, J. Margiewicz, M. Malik, F. Rivera, Z. Zhu, and R. W. Boyd, “Scan-free direct measurement of an extremely high-dimensional photonic state”, *Optica*, 2(4):388–392, April 2015. <https://opg.optica.org/optica/abstract.cfm?uri=optica-2-4-388>, doi:10.1364/OPTICA.2.000388
- [267] D. H. Mahler, L. Rozema, K. Fisher, L. Vermeyden, K. J. Resch, H. M. Wiseman, and A. Steinberg, “Experimental nonlocal and surreal Bohmian trajectories”, *Science Advances*, February 2016. <https://www.science.org/10.1126/sciadv.1501466>, doi:10.1126/sciadv.1501466
- [268] G. S. Thekkadath, L. Giner, Y. Chalich, M. J. Horton, J. Bunker, and J. S. Lundeen, “Direct Measurement of the Density Matrix of a Quantum System”, *Physical Review Letters*, 117(12):120401, September 2016. <https://link.aps.org/doi/10.1103/PhysRevLett.117.120401>, doi:10.1103/PhysRevLett.117.120401
- [269] F. Piacentini, A. Avella, M. P. Levi, M. Gramegna, G. Brida, I. P. Degiovanni, E. Cohen, R. Lussana, F. Villa, A. Tosi, F. Zappa, and M. Genovese, “Measuring Incompatible Observables by Exploiting Sequential Weak Values”, *Physical Review Letters*, 117(17):170402, October 2016. <https://link.aps.org/doi/10.1103/PhysRevLett.117.170402>, doi:10.1103/PhysRevLett.117.170402
- [270] M. Hallaji, A. Feizpour, G. Dmochowski, J. Sinclair, and A. M. Steinberg, “Weak-value amplification of the nonlinear effect of a single photon”, *Nature Physics*, 13(6):540–544, June 2017. <http://www.nature.com/articles/nphys4040>, doi:10.1038/nphys4040
- [271] Y. Kim, Y.-S. Kim, S.-Y. Lee, S.-W. Han, S. Moon, Y.-H. Kim, and Y.-W. Cho, “Direct quantum process tomography via measuring sequential weak values of incompatible observables”, *Nature Communications*, 9(1):192, January 2018. <https://www.nature.com/articles/s41467-017-02511-2>, doi:10.1038/s41467-017-02511-2

- [272] G. Nirala, S. N. Sahoo, A. K. Pati, and U. Sinha, “Measuring average of non-Hermitian operator with weak value in a Mach-Zehnder interferometer”, *Physical Review A*, 99(2):022111, February 2019. <https://link.aps.org/doi/10.1103/PhysRevA.99.022111>, doi:10.1103/PhysRevA.99.022111
- [273] G. J. Pryde, J. L. O’Brien, A. G. White, S. D. Bartlett, and T. C. Ralph, “Measuring a Photonic Qubit without Destroying It”, *Physical Review Letters*, 92(19):190402, May 2004. <https://link.aps.org/doi/10.1103/PhysRevLett.92.190402>, doi:10.1103/PhysRevLett.92.190402
- [274] G. J. Pryde, J. L. O’Brien, A. G. White, T. C. Ralph, and H. M. Wiseman, “Measurement of Quantum Weak Values of Photon Polarization”, *Physical Review Letters*, 94(22):220405, June 2005. <https://link.aps.org/doi/10.1103/PhysRevLett.94.220405>, doi:10.1103/PhysRevLett.94.220405
- [275] S. Wu, “State tomography via weak measurements”, *Scientific Reports*, 3(1):1193, February 2013. <https://www.nature.com/articles/srep01193>, doi:10.1038/srep01193
- [276] B. E. Y. Svensson, “Pedagogical Review of Quantum Measurement Theory with an Emphasis on Weak Measurements”, *Quanta*, 2(1):18–49, May 2013. <http://quanta.ws/ojs/index.php/quanta/article/view/12>, doi:10.12743/quanta.v2i1.12
- [277] B. Tamir and E. Cohen, “Introduction to Weak Measurements and Weak Values”, *Quanta*, 2(1):7–17, May 2013. <http://quanta.ws/ojs/index.php/quanta/article/view/14>, doi:10.12743/quanta.v2i1.14
- [278] J. Dressel, M. Malik, F. M. Miatto, A. N. Jordan, and R. W. Boyd, “Colloquium: Understanding quantum weak values: Basics and applications”, *Reviews of Modern Physics*, 86(1):307–316, March 2014. <https://link.aps.org/doi/10.1103/RevModPhys.86.307>, doi:10.1103/RevModPhys.86.307
- [279] A. Botero, “Weak value tomography of strong quantum measurements”, *Quantum Studies: Mathematics and Foundations*, 5(3):423–437, September 2018. <https://doi.org/10.1007/s40509-018-0155-0>, doi:10.1007/s40509-018-0155-0
- [280] M. Zych, F. Costa, I. Pikovski, and Č. Brukner, “Quantum interferometric visibility as a witness of general relativistic proper time”, *Nature Communications*, 2:505, October 2011. <http://www.nature.com/articles/ncomms1498>, doi:10.1038/ncomms1498

Index

A

`angle` (*Rotation property*), 117
`anticontrols` (*QuantumGate property*), 108
`apply()` (in module `qchronology.mechanics.operations`), 190
`apply()` (*QuantumState method*), 92
`axis` (*Rotation property*), 117

B

`bra()` (in module `qchronology.mechanics.matrices`), 176

C

`coefficient` (*QuantumGate property*), 110
`coefficient()` (in module `qchronology.mechanics.operations`), 192
`coefficient()` (*QuantumState method*), 94
`columnify()` (in module `qchronology.mechanics.operations`), 189
`composite` (*Fourier property*), 131
`conditions` (*QuantumCircuit property*), 146
`conditions` (*QuantumGate property*), 109
`conditions` (*QuantumState property*), 89
`conjugate` (*QuantumGate property*), 109
`conjugate` (*QuantumState property*), 89
`controls` (*QuantumGate property*), 108

D

`dagger()` (in module `qchronology.mechanics.operations`), 189
`dagger()` (*QuantumState method*), 92
`DCTC` (class in `qchronology.quantum.prescriptions`), 164
`dctc_respecting()` (in module `qchronology.quantum.prescriptions`), 163
`dctc_violating()` (in module `qchronology.quantum.prescriptions`), 163
`debug` (*QuantumState property*), 89
`decode()` (in module `qchronology.mechanics.matrices`), 180
`decode_fast()` (in module `qchronology.mechanics.matrices`), 180
`decode_multiple()` (in module `qchronology.mechanics.matrices`), 181
`decode_slow()` (in module `qchronology.mechanics.matrices`), 179
`densify()` (in module `qchronology.mechanics.operations`), 189
`densify()` (*QuantumState method*), 91
`Diagonal` (class in `qchronology.quantum.gates`), 119
`diagram()` (*QuantumCircuit method*), 151
`diagram()` (*QuantumCTC method*), 162
`diagram()` (*QuantumGate method*), 111

`diagram()` (*QuantumState method*), 90
`dim` (*QuantumCircuit property*), 146
`dim` (*QuantumGate property*), 108
`dim` (*QuantumState property*), 88
`distance()` (in module `qchronology.mechanics.quantities`), 184
`distance()` (*QuantumState method*), 99

E

`encode()` (in module `qchronology.mechanics.matrices`), 178
`entries` (*Diagonal property*), 121
`entropy()` (in module `qchronology.mechanics.quantities`), 185
`entropy()` (*QuantumState method*), 101
`exponent` (*QuantumGate property*), 109
`exponentiation` (*Diagonal property*), 121

F

`family` (*QuantumGate property*), 110
`family` (*QuantumState property*), 89
`fidelity()` (in module `qchronology.mechanics.quantities`), 184
`fidelity()` (*QuantumState method*), 100
`form` (*QuantumState property*), 88
`Fourier` (class in `qchronology.quantum.gates`), 129
`free_symbol` (*DCTC property*), 165

G

`gate()` (*QuantumCircuit method*), 148
`gate_is_linear` (*QuantumCircuit property*), 147
`GateInterleave` (class in `qchronology.quantum.gates`), 133
`gates` (*GateInterleave property*), 135
`gates` (*QuantumCircuit property*), 145
`GateStack` (class in `qchronology.quantum.gates`), 135
`GellMann` (class in `qchronology.quantum.gates`), 115

H

`Hadamard` (class in `qchronology.quantum.gates`), 127

I

`index` (*GellMann property*), 116
`index` (*Pauli property*), 114
`input()` (*QuantumCircuit method*), 147
`input()` (*QuantumCTC method*), 161
`input_is_vector` (*QuantumCircuit property*), 147
`inputs` (*QuantumCircuit property*), 145
`is_vector` (*QuantumState property*), 90

K

`ket()` (*in module qhronology.mechanics.matrices*), 175
`kind` (*QuantumState property*), 88

L

`label` (*QuantumGate property*), 110
`label` (*QuantumState property*), 89

M

`matrix` (*DCTC property*), 165
`matrix` (*PCTC property*), 170
`matrix` (*QuantumCircuit property*), 147
`matrix` (*QuantumGate property*), 110
`matrix` (*QuantumState property*), 90
`MatrixState` (*class in qhronology.quantum.states*), 103
`measure()` (*in module qhronology.mechanics.operations*), 194
`measure()` (*QuantumCircuit method*), 150
`measure()` (*QuantumState method*), 95
`Measurement` (*class in qhronology.quantum.gates*), 131
`merge` (*GateInterleave property*), 135
`MixedState` (*class in qhronology.quantum.states*), 104
`mutual()` (*in module qhronology.mechanics.quantities*), 186
`mutual()` (*QuantumState method*), 102

N

`norm` (*QuantumState property*), 89
`normalize()` (*in module qhronology.mechanics.operations*), 192
`normalize()` (*QuantumState method*), 93
`Not` (*class in qhronology.quantum.gates*), 125
`notation` (*QuantumGate property*), 110
`notation` (*QuantumState property*), 89
`num_systems` (*QuantumCircuit property*), 146
`num_systems` (*QuantumGate property*), 108
`num_systems` (*QuantumState property*), 90
`num_systems_gates` (*QuantumCircuit property*), 147
`num_systems_gross` (*QuantumCircuit property*), 147
`num_systems_inputs` (*QuantumCircuit property*), 146
`num_systems_net` (*QuantumCircuit property*), 147
`num_systems_removed` (*QuantumCircuit property*), 147

O

`observable` (*Measurement property*), 133
`OperationsMixin` (*class in qhronology.mechanics.operations*), 197
`operators` (*Measurement property*), 133
`output()` (*DCTC method*), 166
`output()` (*PCTC method*), 171

`output()` (*QuantumCircuit method*), 149
`output()` (*QuantumGate method*), 110
`output()` (*QuantumState method*), 90
`output_is_vector` (*QuantumCircuit property*), 147
`output_respecting()` (*DCTC method*), 165
`output_respecting()` (*PCTC method*), 170
`output_violating()` (*DCTC method*), 165
`output_violating()` (*PCTC method*), 170

P

`partial_trace()` (*in module qhronology.mechanics.operations*), 193
`partial_trace()` (*QuantumState method*), 94
`Pauli` (*class in qhronology.quantum.gates*), 112
`PCTC` (*class in qhronology.quantum.prescriptions*), 169
`pctc_respecting()` (*in module qhronology.quantum.prescriptions*), 169
`pctc_violating()` (*in module qhronology.quantum.prescriptions*), 168
`Phase` (*class in qhronology.quantum.gates*), 117
`phase` (*Phase property*), 119
`post_is_vector` (*QuantumCircuit property*), 147
`postselect()` (*in module qhronology.mechanics.operations*), 196
`postselect()` (*QuantumState method*), 97
`postselections` (*QuantumCircuit property*), 145
`PureState` (*class in qhronology.quantum.states*), 104
`purity()` (*in module qhronology.mechanics.quantities*), 183
`purity()` (*QuantumState method*), 98

Q

`QuantitiesMixin` (*class in qhronology.mechanics.quantities*), 187
`quantum_state()` (*in module qhronology.mechanics.matrices*), 177
`QuantumCircuit` (*class in qhronology.quantum.circuits*), 139
`QuantumCTC` (*class in qhronology.quantum.prescriptions*), 157
`QuantumGate` (*class in qhronology.quantum.gates*), 105
`QuantumState` (*class in qhronology.quantum.states*), 83

R

`reset()` (*QuantumState method*), 91
`reverse` (*Fourier property*), 131
`rewrite()` (*in module qhronology.mechanics.operations*), 191
`rewrite()` (*QuantumState method*), 93
`Rotation` (*class in qhronology.quantum.gates*), 116

S

`shift` (*Summation property*), 125

simplify() (in module *qhronology.mechanics.operations*), 190
simplify() (*QuantumState* method), 92
spec (*QuantumGate* property), 108
spec (*QuantumState* property), 88
state() (*QuantumCircuit* method), 149
state_respecting() (*DCTC* method), 167
state_respecting() (*PCTC* method), 172
state_violating() (*DCTC* method), 166
state_violating() (*PCTC* method), 171
Summation (class in *qhronology.quantum.gates*), 124
Swap (class in *qhronology.quantum.gates*), 121
symbols (*QuantumCircuit* property), 145
symbols (*QuantumGate* property), 108
symbols (*QuantumState* property), 88
systems (*QuantumCircuit* property), 146
systems (*QuantumGate* property), 110
systems (*QuantumState* property), 90
systems_postselections (*QuantumCircuit* property), 146
systems_removed (*QuantumCircuit* property), 146
systems_respecting (*QuantumCTC* property), 161
systems_traces (*QuantumCircuit* property), 146
systems_violating (*QuantumCTC* property), 161

T

targets (*QuantumGate* property), 108
trace() (in module *qhronology.mechanics.quantities*), 183
trace() (*QuantumState* method), 98
traces (*QuantumCircuit* property), 145

V

vector_basis() (in module *qhronology.mechanics.matrices*), 175
VectorState (class in *qhronology.quantum.states*), 103

