# 7 Quantum Error Correction

In this section, we discuss techniques for quantum error correction, which is an absolute necessity for the success of quantum computing, given the high likelihood of noise, errors, and decoherence in larger circuits. We have ignored this topic so far and assumed an ideal, error-free execution environment. For real machines, this assumption will not hold. Quantum error correction is a fascinating and wide-ranging topic. This section is mostly an introduction, with focus on just a few core principles.

## 7.1 Quantum Noise

Building a real, physical quantum computer big enough to perform useful computation presents an enormous challenge. On one hand, the quantum system must be isolated from the environment as much as possible to avoid entanglement with the environment and other perturbations, which may introduce errors. For example, molecules may bump into qubits and change their relative phase, even at temperatures of close to absolute zero. On the other hand, the quantum system cannot be entirely isolated because we want to program the machine, perhaps dynamically, and make measurements.

Here is a summary of available technologies, as presented in Nielsen and Chuang (2011). Table 7.1 shows the underlying technology, the time $\tau_Q$ the system may stay coherent before it starts entangling with the environment, the time $\tau_{op}$ it takes to apply a typical unitary gate, and the number $n_{op}$ of operations that can be executed while still in a coherent state.

For several technologies, the number of coherently executable instructions is rather small and won't suffice to execute very large algorithms with potentially billions of gates.

Errors are inevitable, given the quantum scale and very high likelihood of the environment perturbing the system. To compare the expected quantum and classical error rates – for a modern CPU, a typical error rate is about one per year, or one error for every $10^{17}$ operations. The actual error rate might be higher, but mitigation strategies are in place. In contrast, data from 2020 from IBM shows an average single-qubit gate error rate of about one per $10^{-3}$ seconds, and one per $10^{-2}$ seconds for two-qubit gates. Based on frequency, this could reach up to one error for every 200 operations. This is a difference of almost 10 orders of magnitude!

**Table 7.1** Estimates for decoherence times (secs), gate application latency (secs), and number of gates that can be applied while coherent. Data from Nielsen and Chuang (2011).

| System | $\tau_Q$ | $\tau_{op}$ | $n_{op}$ |
|---|---|---|---|
| Nuclear spin | $10^{-2} - 10^{-8}$ | $10^{-3} - 10^{-6}$ | $10^5 - 10^{14}$ |
| Electron spin | $10^{-3}$ | $10^{-7}$ | $10^4$ |
| Ion trap | $10^{-1}$ | $10^{-14}$ | $10^{13}$ |
| Electron - Au | $10^{-8}$ | $10^{-14}$ | $10^6$ |
| Electron - GaAs | $10^{-10}$ | $10^{-13}$ | $10^3$ |
| Quantum dot | $10^{-6}$ | $10^{-9}$ | $10^3$ |
| Optical cavity | $10^{-5}$ | $10^{-14}$ | $10^9$ |
| Microwave cavity | $10^0$ | $10^{-4}$ | $10^4$ |

What are possible error conditions, and how do we model the likelihood of their occurrence?

## Bit-Flip Error

The *bit-flip error* causes the probability amplitudes of a qubit to flip, similar to the effect of an X-gate:

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \beta|0\rangle + \alpha|1\rangle.$$

This is also called a *dissipation-induced* bit-flip error. Dissipation is the process of losing energy to the environment. If we think of a qubit in state $|1\rangle$ as an electron's excited state, as it loses energy, it may fall to the lower energy $|0\rangle$ state and emit a photon. Correspondingly, it may jump from $|0\rangle$ to $|1\rangle$ by absorbing a photon (in which case it should probably be called an *excitation-induced* error).

## Phase-Flip Error

The *phase-flip error* causes the relative phase to flip from $+1$ to $-1$, similar to the effect of a Z-gate:

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \alpha|0\rangle - \beta|1\rangle.$$

This is also called a *decoherence-induced* phase shift error. In the example, we shifted the phase by $\pi$, but for decoherence we should also consider much smaller phase changes and their insidious tendency to compound over time.

## Combined Phase/Bit-Flip Error

This is the combination of the two error conditions above:

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \beta|0\rangle - \alpha|1\rangle.$$

This is equivalent to applying the Y-gate, as we've seen before, ignoring the global phase:

$$Y\big(\alpha|0\rangle + \beta|1\rangle\big) = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = -i\beta|0\rangle + i\alpha|1\rangle = -i\big(\beta|0\rangle - \alpha|1\rangle\big).$$

### No Error

We should mention this one for completeness; it's the equivalent effect of applying an identity gate to a qubit, or, equally, doing nothing.

These errors will occur with a certain probability. To model this properly, we will introduce the concept of quantum operations next, which allow the formalizing of the statistical distribution of error conditions in an elegant way.

## 7.1.1 Quantum Operations

So far we have mostly focused on describing quantum states as vectors of probability amplitudes. We indicated that states can also be described with density operators, which allow describing mixtures of states. In the following, we adopt the formalism proposed in Nielsen and Chuang (2011).

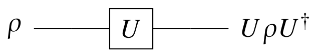
Similar to how a state evolves with $|\psi'\rangle = U|\psi\rangle$, a state's density operator $\rho = |\psi\rangle\langle\psi|$ evolves as:

$$\rho' = \mathcal{E}(\rho).$$

Where the $\mathcal{E}$ is called a *quantum operation*. The two types of operations discussed in this book are unitary transformations and measurements (note the matrix multiplication from both sides):

$$\mathcal{E}(\rho) = U\rho U^\dagger \quad \text{and} \quad \mathcal{E}_M(\rho) = M\rho M^\dagger. \tag{7.1}$$

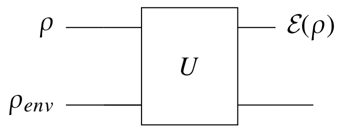
In a *closed* quantum system, which has no interaction with the environment, the system evolves as:



In an *open* system, we model the system as the tensor product of state and environment as $\rho \otimes \rho_{env}$. The system evolves as described in Equation (7.1) as:

$$U(\rho \otimes \rho_{env})\,U^\dagger.$$

We can visualize this with this conceptual circuit:



To describe the system without the environment, we trace out the environment using the methodology from Section 2.14:

$$\mathcal{E}(\rho) = \operatorname{tr}_{env}\left[U(\rho \otimes \rho_{env})\,U^{\dagger}\right]. \tag{7.2}$$

Now, quantum operators can be expressed in the *operator-sum representation*, which describes the behavior of the principle system only, based on Equation (7.2). Let $|e_k\rangle$ be the orthonormal basis of the environment and $e_{env} = |e_0\rangle\langle e_0|$ the environment's initial state. It can be shown (see Nielsen and Chuang, 2011, section 8.2.3) that:

$$\mathcal{E}(\rho) = \sum_k E_k \rho E_k^{\dagger},$$

where $E_k = \langle e_k|U|e_0\rangle$. The $E_k$ are the *operation elements* for the quantum operation $\mathcal{E}$. They are also called *Krauss operators*[1] and operate on the principal system only. Now let's see how we can use this formalism to describe the various error modes.

### 7.1.2 Bit Flip and Phase Flip Channels

The term *channel* is an abstraction in information theory to model noise and errors. It assumes that information must be conveyed from a source to a destination *somehow*. That "somehow" is commonly described as a channel. We can use this to describe the above error modes in the following way.

The *bit-flip channel* flips the states from $|0\rangle$ to $|1\rangle$ and vice versa with probability $1 - p$. It has the operation elements:

$$E_0 = \sqrt{p}\,I = \sqrt{p}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \quad \text{and} \quad E_1 = \sqrt{1-p}\,X = \sqrt{1-p}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}.$$

The *phase-flip channel* flips the phase as described above with probability $1 - p$. It has the operation elements:

$$E_0 = \sqrt{p}\,I = \sqrt{p}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \quad \text{and} \quad E_1 = \sqrt{1-p}\,Z = \sqrt{1-p}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}.$$

Finally, the *bit-flip phase-flip channel* has these operation elements:

$$E_0 = \sqrt{p}\,I = \sqrt{p}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \quad \text{and} \quad E_1 = \sqrt{1-p}\,Y = \sqrt{1-p}\begin{bmatrix}0 & -i\\ i & 0\end{bmatrix}.$$

[1] This notation is sloppy, as $U$ applies to both environment and state. Because this detail is not essential in our context, we tolerate it.

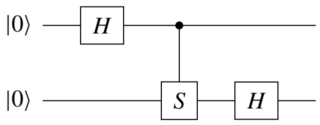


**Figure 7.1** Circuit before noise injection.

## 7.1.3 Depolarization Channel

The *depolarization channel* is another standard way to describe quantum noise. Depolarization means that an original state is transformed into a completely mixed state $I/2$. We only briefly talked about pure and mixed states in Section 2.14, but in short, a maximally mixed state of $I/2$ means that the state is maximally entangled with something else, for example, the environment.

*Quantum noise* means that a state remains unmodified with probability $1 - p$. The state, expressed as its density matrix $\rho$, becomes the following in the presence of noise:

$$\rho' = p\frac{I}{2} + (1-p)\rho.$$

For an arbitrary $\rho$, it can be shown that the following holds in operator-sum notation (see also the test `test_rho` in file `lib/ops_test.py`). This equation is related to Equation (2.5).

$$\frac{I}{2} = \frac{\rho + X\rho X + Y\rho Y + Z\rho Z}{4}$$

Suppose we assign a probability of $(1 - p)$ for a state to remain unmodified by noise, and we assign a probability of $1/3$ for each of the operators X, Y, and Z to introduce noise (other probability distributions are possible). In that case, the operator sum expression above can be transformed into:

$$\mathcal{E}(\rho) = (1-p)\rho + \frac{p}{3}(X\rho X + Y\rho Y + Z\rho Z).$$

This is the result we were looking for. It allows us to model quantum noise by simply injecting Pauli gates with a given probability. Assume a gate $E$, which may be either of the Pauli matrices with probability as follows:

$$E = \begin{cases} X & \text{with } p_x, \\ Y & \text{with } p_y, \\ Z & \text{with } p_z, \\ I & \text{with } 1-(p_x+p_y+p_z). \end{cases}$$

To model noise, we introduce error gates $E$ with a given probability, injecting bit-flip and phase-flip errors. An example circuit before and after error injection is shown in Figures 7.1 and 7.2, respectively. It is very educational to inject these error gates and evaluate their impact on various algorithms. We will do just that in Section 7.2.

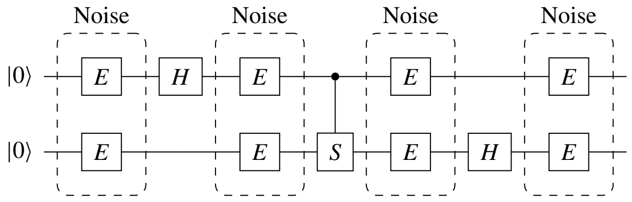


**Figure 7.2** Circuit after noise injection.

## 7.1.4 Amplitude and Phase Damping

We mention amplitude damping and phase damping for completeness, but we will not elaborate further.

*Amplitude damping* seeks to model *energy dissipation*, the energy loss in a quantum system. It is described with these two operator elements, with $\gamma$ (gamma) being the likelihood of energy loss, such as the emission of a photon in a physical system:

$$E_0 = \begin{bmatrix} 1 & 0 \\ 0 & \sqrt{1-\gamma} \end{bmatrix} \quad \text{and} \quad E_1 = \begin{bmatrix} 0 & \sqrt{\gamma} \\ 0 & 0 \end{bmatrix}.$$

*Phase damping* describes the process of a system losing relative phase between qubits, thus introducing errors in algorithms that rely on successful quantum interference. The operator elements are:

$$E_0 = \begin{bmatrix} 1 & 0 \\ 0 & \sqrt{1-\gamma} \end{bmatrix} \quad \text{and} \quad E_1 = \begin{bmatrix} 0 & 0 \\ 0 & \sqrt{\gamma} \end{bmatrix}.$$

Note that the factor $\gamma$ might be expressed as an exponential function in more realistic modeling environments.

## 7.1.5 Imprecise Gates

Gates themselves may not be perfect. There could be issues with manufacturing, external influences, temperature, and other conditions, all influencing gate accuracy. Additionally, it is unlikely that all software gates we use in this text will be available on physical machines. Software gates will have to be decomposed into hardware gates or be approximated. Approximations have residual errors, as we detailed in Section 6.15 on the Solovay–Kitaev algorithm.

The impact of gate imprecision varies by algorithm. Since we wrote the algorithms in source, we can run experiments and inject various error distributions. In the following brief example, we modify the final inverse QFT in the phase estimation circuit by introducing errors in the $R_k$ phase gates. To achieve this, we compute a random number in the range from 0.0 to 1.0 and scale a noise factor $n_f$ with it. As an example, a factor of $n_f = 0.1$ means a *maximum* error of 10% can be introduced. The actual values will range randomly from 0% to the limit of 10%. This is a very simple model, but you can experiment with other error distributions.

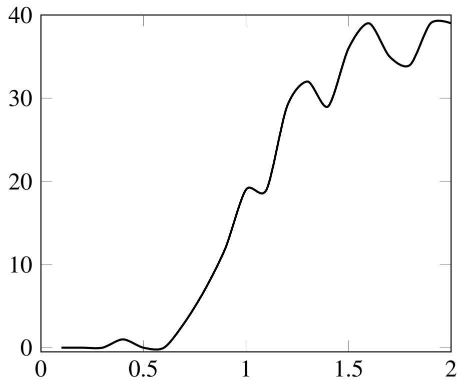


**Figure 7.3** Phase estimation errors exceeding a threshold of 2% from increasing levels of noise, for N = 50 experiments per setting. The x-axis represents noise ranging from 0% to 200%, the y-axis represents the number of experiments exceeding the threshold.

```
def Rk(k):
  return Operator(np.array([(1.0, 0.0),
       (0.0, cmath.exp((1 + (random.random() * flags.FLAGS.noise)) *
       (2.0 * cmath.pi * 1j / 2**k)))]))
```

Then, for values of $n_f$ ranging from 0.0 to 2.0, we run 50 experiments and count the number of experiments that result in a phase estimation error larger than 2%. Hence, we test the robustness of phase estimation against small to large errors in the inverse QFT rotation gates. Figure 7.3 shows the distribution.

We see that the inverse QFT is surprisingly robust against sizeable maximum errors in the rotation gates, but this is just an anecdote. The exact outcome would depend on the statistical distribution of the actual errors. We should also expect that each algorithm has different tolerances and sensitivities. For comparison, introducing depolarization with just 0.1% probability leads to significantly different outcomes in the order finding algorithm, which is very sensitive to this particular type of error.

## 7.2 Quantum Error Correction

We will need some form of error correction techniques to control the impact of noise. In classical computing, a large body of known error correction techniques exists. Error correction code memory, or ECC (Wikipedia, 2021b), may be one of the best known ones. There are many more techniques that prevent invalid data, missing data, or spurious data. NASA, in particular, has developed impressive techniques to communicate with their ever-more-distant exploratory vehicles.

A simple classical error correction technique is based on repetition codes and majority voting. For example, we could triple each binary digit:

**Table 7.2** Majority voting for a simple repetition code.

| Measured | Voted | Measured | Voted |
|---|---|---|---|
| 000 | 0 | 111 | 1 |
| 001 | 0 | 110 | 1 |
| 010 | 0 | 101 | 1 |
| 100 | 0 | 011 | 1 |

$$0 \rightarrow 000$$
$$1 \rightarrow 111$$

As we receive data over a noisy channel, we measure it and perform majority voting with the scheme shown in Table 7.2. This simple scheme does not account for missing or erroneous bits, but it is good enough to explain the basic principles.

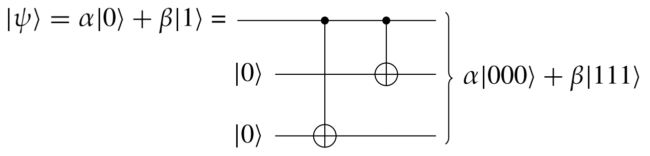
In quantum computing, the situation is generally more difficult:

- Physical quantum computers operate at the quantum level of atomic spins, photons, and electrons. There is a very high probability of encountering errors or decoherence, especially for longer-running computations.
- Errors can be more subtle than simple bit flips. There are multiple error modes.
- Errors, such as relative phase errors, compound during execution.
- Simple repetition codes will not work because of the no-cloning theorem.
- Most problematically, you cannot observe errors, as that would constitute a measurement that destroys the superposition and entanglement that algorithms rely upon.

Because of these difficulties, especially because of the inability to read a corrupted state, early speculation was that error correction code could not exist. Hence, it would be nearly impossible ever to produce a viable quantum computer (Haroche and Raimond, 1996; Rolf, 1995). This, fortunately, changed when Shor presented a viable nine-qubit error correction code (Shor, 1995). The principles of this approach underlie many quantum error correction techniques today.

## 7.2.1 Quantum Repetition Code

This circuit can be used to produce the quantum repetition code. Note its similarity to the GHZ circuit from Section 2.11.4:


$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ =
$|0\rangle$
$|0\rangle$
$\alpha|000\rangle + \beta|111\rangle$

In code, using a random qubit for demonstration:

```
    qbit = state.qubit(random.random())
    psi = qbit * state.zeros(2)
    psi = ops.Cnot(0, 2)(psi)
    psi = ops.Cnot(0, 1)(psi)
    psi.dump()
>>
|000> (|0>):  ampl: +0.78+0.00j prob: 0.61 Phase:   0.0
|111> (|7>):  ampl: +0.62+0.00j prob: 0.39 Phase:   0.0
```

Note again that these states do not violate the no-cloning theorem as we are not constructing $(\alpha|0\rangle + \beta|1\rangle)^{\otimes 3}$.

## 7.2.2 Correct Bit-Flip errors

Here is the main *trick* to error correction. It is related to quantum teleportation. First we introduce redundancy and triple each qubit into a GHZ state. We entangle this three-qubit state with two ancillae and measure *only* the ancillae, leaving the original state intact. Based on the measurement outcome, we apply gates to the original three-qubit state to correct it.

Figure 7.4 shows this procedure in circuit notation, assuming a single qubit-flip error in qubit 0, which we indicate on the left side of the circuit. The state $|\psi_1\rangle$ right before a measurement is this, where the bottom two qubits have been flipped to $|10\rangle$:

$$|\psi_1\rangle = \alpha|10010\rangle + \beta|01110\rangle,$$

which, right after measurement, turns into:

$$|\psi_2\rangle = \big(\alpha|100\rangle + \beta|011\rangle\big) \otimes |10\rangle.$$

The measurement outcome is called the *error syndrome*. Based on the syndrome, we know what to do next and which qubit to flip back with another X-gate.

- For a measurement result of $|00\rangle$, do nothing.
- For a measurement result of $|01\rangle$, apply X-gate to qubit 2.
- For a measurement result of $|10\rangle$, apply X-gate to qubit 0.
- For a measurement result of $|11\rangle$, apply X-gate to qubit 1.

The way Figure 7.4 is drawn is a bit sloppy because the function of gate $R$ is different for each measurement outcome. Making physical measurements and reacting to the outcome is not a realistic scenario; it would be hard to achieve in practice, and even if it did, it would likely destroy a quantum computer's performance advantage because of Amdahl's law.[2] In larger circuits, we should also make sure to disentangle the ancillae.

[2] https://en.wikipedia.org/wiki/Amdahl%27s_law.

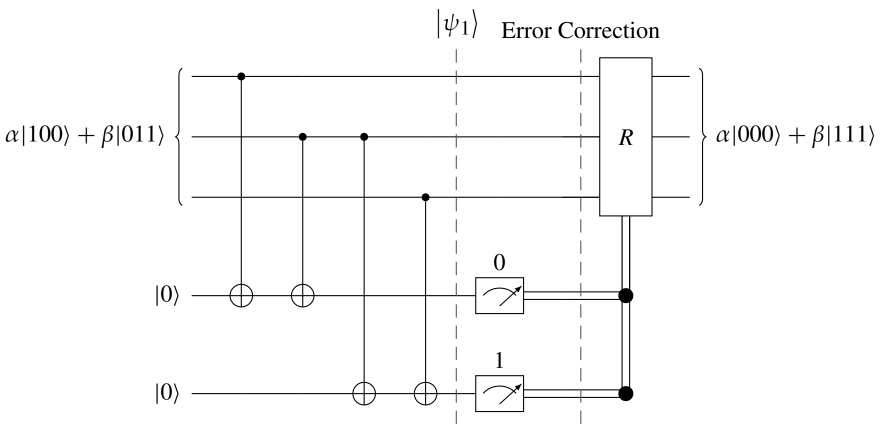

$|\psi_1\rangle$
Error Correction
$\alpha|100\rangle + \beta|011\rangle$
$\alpha|000\rangle + \beta|111\rangle$
R
$|0\rangle$
$|0\rangle$
0
1

**Figure 7.4** Bit-flip error correcting circuit.

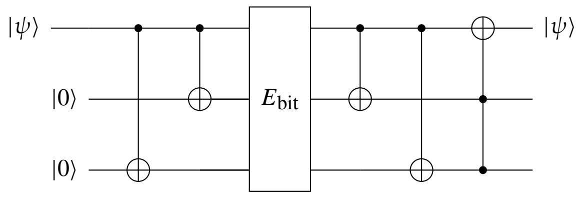


**Figure 7.5** Error correction for bit-flip error.

A common construction to correct bit-flip errors can be found in the circuit in Figure 7.5. The noisy channel $E$, introduces errors according to Equation (7.3), the bit-flip error:

$$\varepsilon(\rho) = (1 - p)\rho + p(X\rho X). \tag{7.3}$$

In code, we can inject an error by introducing an X-gate, as in the following snippet:

```
def test_x_error(self):
  qc = circuit.qc('x-flip / correction')
  qc.qubit(0.6)

  # Replication code setup.
  qc.reg(2, 0)
  qc.cx(0, 2)
  qc.cx(0, 1)
  qc.psi.dump('after setup')

  # Error insertion.
  qc.x(0)

  # Fix.
```

```
    qc.cx(0, 1)
    qc.cx(0, 2)
    qc.ccx(1, 2, 0)
    qc.psi.dump('after correction')
```

If no error has been injected, we will see this output:

```
|210> 'after setup'
|000> (|0>):  ampl: +0.60+0.00j prob: 0.36 Phase:   0.0
|111> (|7>):  ampl: +0.80+0.00j prob: 0.64 Phase:   0.0
|210> 'after correction'
|000> (|0>):  ampl: +0.60+0.00j prob: 0.36 Phase:   0.0
|100> (|4>):  ampl: +0.80+0.00j prob: 0.64 Phase:   0.0
```

If an error has indeed been injected, the state becomes:

```
|210> 'after setup'
|000> (|0>):  ampl: +0.60+0.00j prob: 0.36 Phase:   0.0
|111> (|7>):  ampl: +0.80+0.00j prob: 0.64 Phase:   0.0
|210> 'after correction'
|011> (|3>):  ampl: +0.60+0.00j prob: 0.36 Phase:   0.0
|111> (|7>):  ampl: +0.80+0.00j prob: 0.64 Phase:   0.0
```

Note the small difference in the final states with nonzero probabilities. In the case without an injected error, the state resumes to $|000\rangle$ and $|100\rangle$, the original input state. In the case with an injected error, the ancilla qubits are $|11\rangle$ for both resulting states.

## 7.2.3 Correct Phase-Flip Errors

We can use the same idea to correct phase-flip errors. Remember that applying uniform Hadamard gates puts a state in the Hadamard basis. A phase-flip error in the computational basis is the same as a bit-flip error in the Hadamard basis.

Correspondingly, we can use the circuit in Figure 7.6 to create a quantum repetition and error correction circuit, similar to Figure 7.5, but with surrounding Hadamard gates.

We use a similar code sequence for this as above, but we change it to the following for error injection:

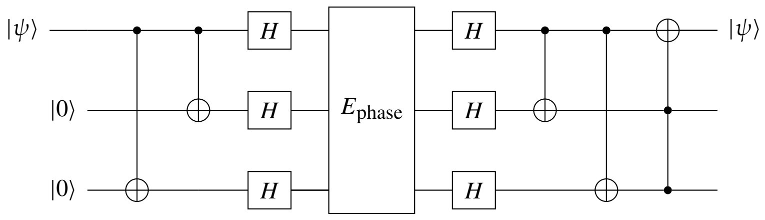


**Figure 7.6** Error correction for phase-flip error.

```
    [...]
    qc.h(0)
    qc.h(1)
    qc.h(2)

    qc.z(0)

    qc.h(0)
    qc.h(1)
    qc.h(2)
    [...]
```

The probability distribution of the resulting nonzero probability states is the same, but we get a few states with phases. For example, without error injection:

```
|210> 'after setup'
|000> (|0>):  ampl: +0.60+0.00j prob: 0.36 Phase:   0.0
|111> (|7>):  ampl: +0.80+0.00j prob: 0.64 Phase:   0.0
|210> 'after correction'
|000> (|0>):  ampl: +0.60+0.00j prob: 0.36 Phase:   0.0
|001> (|1>):  ampl: +0.00+0.00j prob: 0.00 Phase:   0.0
|010> (|2>):  ampl: -0.00+0.00j prob: 0.00 Phase: 180.0
|011> (|3>):  ampl: -0.00+0.00j prob: 0.00 Phase: 180.0
|100> (|4>):  ampl: +0.80+0.00j prob: 0.64 Phase:   0.0
|101> (|5>):  ampl: +0.00+0.00j prob: 0.00 Phase:   0.0
|110> (|6>):  ampl: +0.00+0.00j prob: 0.00 Phase:   0.0
```

## 7.3 Nine-Qubit Shor Code

All of this leads up to the final nine-qubit Shor code (Shor, 1995), which is a combination of the above. It combines the circuits to find bit-flip, phase-flip, and combined errors into one large circuit, as shown in Figure 7.7.

The Shor nine-qubit circuit is able to identify and correct one bit-flip error, one phase-flip error, or one of each on any of the nine qubits! Let's verify this in code and apply all Pauli gates to each of the qubits of this circuit. For this experiment, we construct a qubit with $\alpha = 0.60$ (the code can be found in file `lib/circuit_test.py`):

```
  def test_shor_9_qubit_correction(self):
    for i in range(9):
      qc = circuit.qc('shor-9')
      print(f'Init qubit as 0.6|0> + 0.8|1>, error on qubit {i}')
      qc.qubit(0.6)
      qc.reg(8, 0)

      # Left Side.
      qc.cx(0, 3)
```

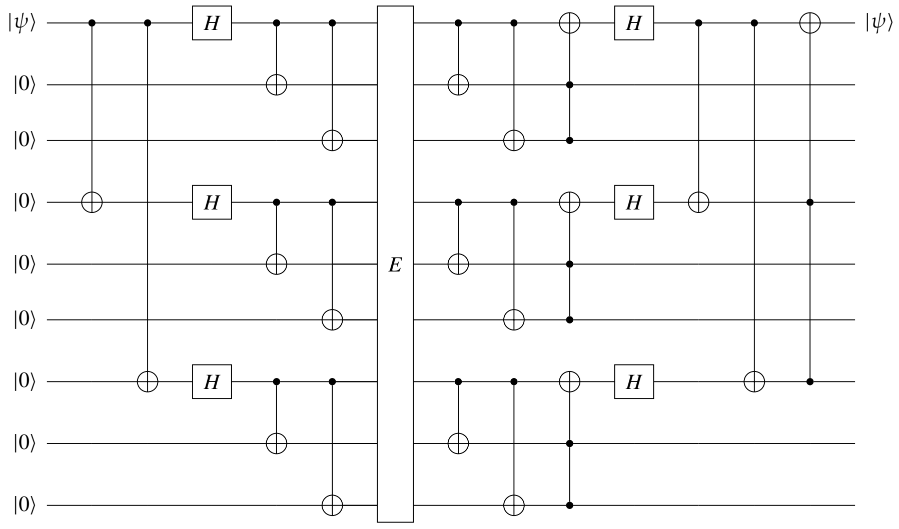


**Figure 7.7** Shor’s nine-qubit error correction circuit.

```
qc.cx(0, 6)
qc.h(0); qc.h(3); qc.h(6);
qc.cx(0, 1); qc.cx(0, 2)
qc.cx(3, 4); qc.cx(3, 5)
qc.cx(6, 7); qc.cx(6, 8)

# Error insertion, use x(i), y(i), or z(i)
qc.x(i)

# Fix.
qc.cx(0, 1); qc.cx(0, 2); qc.ccx(1, 2, 0)
qc.h(0)
qc.cx(3, 4); qc.cx(3, 5); qc.ccx(4, 5, 3)
qc.h(3)
qc.cx(6, 7); qc.cx(6, 8); qc.ccx(7, 8, 6)
qc.h(6)

qc.cx(0, 3); qc.cx(0, 6)
qc.ccx(6, 3, 0)

prob0, s = qc.measure_bit(0, 0)
prob1, s = qc.measure_bit(0, 1)
print('          Measured: {:.2f}|0> + {:.2f}|1>'.format(
    math.sqrt(prob0), math.sqrt(prob1)))
```

Indeed, we get the desired result:

```
Initialize qubit as 0.60|0> + 0.80|1>, error on qubit 0
          Measured: 0.60|0> + 0.80|1>
[...]
Initialize qubit as 0.60|0> + 0.80|1>, error on qubit 8
          Measured: 0.60|0> + 0.80|1>
```

There are several other techniques and formalisms for quantum information and quantum error correction. We only want to mention a small number of influential works. A good introduction and overview can be found in Devitt et al. (2013). Andrew Steane published the seven-qubit Steane code in Steane (1996). A five-qubit error correction code was discussed by Cory et al. (1998).