

Quantum Error Correction: A Brief Walk Through Stabilizer Codes

ANAGHA G.

October 2023

Contents

1	Introduction	2
2	Literature Survey	2
3	Methodology	2
3.1	Motivation for Stabilizer Formalism	2
3.1.1	Shor's Code	3
3.2	Code Concatenation	3
3.3	Preliminaries	3
3.3.1	Some terminology	3
3.3.2	Group Theory	4
3.3.3	Pauli Group	4
3.3.4	Clifford Groups	4
3.4	An introduction to Stabilizer Codes	4
3.4.1	Stabilizer Group	4
3.4.2	Stabilizer States	4
3.4.3	Check matrix	4
3.4.4	Unitary dynamics	5
3.4.5	Logical Operators	5
3.5	Connections between Stabilizer Codes and Classical Linear Codes	5
3.6	Gottesman–Knill Theorem	6
3.7	On the complexity of decoding quantum stabilizer codes	6
3.8	Importance of nonstabilizer resources for quantum computation	6
3.9	Examples	7
3.9.1	Three-qubit bit-flip code	7
3.9.2	Three-qubit phase-flip code	7
3.9.3	Shor's Code	7
3.9.4	Steane Code	7
3.10	Implementation	8
3.10.1	Construction of Encoding Circuits for Stabilizer Codes	8
3.11	Decoding circuit	10
3.12	Conclusion	12
	References	12

1 Introduction

Quantum error correction is a core concept in quantum information theory. Quantum algorithms exploit coherent interference, which is vulnerable to noise and unwanted interactions with the environment. Quantum error-correcting codes protect information by encoding it into a larger physical system. This report focuses on stabilizer codes, whose description by commuting Pauli operators connects quantum error correction to classical linear coding theory. We also study their computational complexity and construct encoding and decoding circuits for the Steane code using the STAC library.

Classical error correction supplies useful intuition, but the quantum setting introduces additional constraints:

- **No-cloning theorem:** An arbitrary unknown quantum state cannot be copied. Encoding must therefore distribute quantum information through an isometry rather than produce independent copies of the input.
- Measuring physical qubits in the computational basis generally disturbs an encoded superposition. Suitable syndrome measurements, however, reveal information about errors without measuring the encoded amplitudes.
- A qubit can suffer both bit and phase errors, as well as their combinations.

These constraints motivate a description of errors and measurements that preserves the logical information.

2 Literature Survey

Introductory treatments explain the transition from classical redundancy to quantum error correction [2, 13]. Gottesman's thesis develops the stabilizer formalism and methods for encoding and decoding [1]; additional expositions provide complementary examples [6, 7, 9]. Two results are particularly relevant here: Aaronson and Gottesman's efficient simulation algorithm and complexity classification [4], and Iyer and Poulin's hardness result for degenerate maximum-likelihood decoding [3]. The implementation section uses STAC [12] and works through the Steane code with an explicit choice of qubit ordering and logical basis.

3 Methodology

3.1 Motivation for Stabilizer Formalism

The classical three-bit repetition code encodes

$$0 \mapsto 000, \quad 1 \mapsto 111.$$

A single bit flip can change 111 to 101. A *majority* vote then recovers the original bit, whereas two bit flips can lead to an incorrect decision. The same information can be obtained from parity checks. The Hamming distance between two bit strings is the number of positions in which they differ, and the minimum distance d of a classical code is the smallest such distance between distinct codewords. A distance- d code corrects every error pattern of weight at most

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor, \quad \text{equivalently, } d \geq 2t + 1 \text{ is sufficient to correct } t \text{ errors.}$$

The equality $d = 2t + 1$ need not hold, for example when d is even.

For a quantum repetition code, define

$$|0\rangle_L = |000\rangle, \quad |1\rangle_L = |111\rangle.$$

An input $|\psi\rangle = a|0\rangle + b|1\rangle$, with $|a|^2 + |b|^2 = 1$, becomes

$$|\psi\rangle_L = a|000\rangle + b|111\rangle.$$

An X error on the first qubit produces

$$X_1|\psi\rangle_L = a|100\rangle + b|011\rangle.$$

Measuring each qubit would reveal the encoded bit and destroy the coherence between the two logical basis states. Instead, measure Z_1Z_2 and Z_1Z_3 . Both have eigenvalue +1 on the code space. The errors X_1 , X_2 , and X_3 give respective eigenvalue pairs $(-1, -1)$, $(-1, +1)$, and $(+1, -1)$, independent of a and b . Applying the indicated X restores the encoded state, provided at most one bit flip occurred.

This code does not protect against phase errors: Z_2 maps the state to $a|000\rangle - b|111\rangle$ without changing either parity check. Applying $H^{\otimes 3}$ gives a phase-flip repetition code with basis $|+++\rangle$ and $|- - -\rangle$, where $|\pm\rangle = (|0\rangle \pm$

$|1\rangle)/\sqrt{2}$. Its checks X_1X_2 and X_1X_3 detect single Z errors in the same way.

Digitization of errors. The operators I, X, Y, Z form a basis for all single-qubit operators, so an error operator can be expanded as $E = c_I I + c_X X + c_Y Y + c_Z Z$, with $Y = iXZ$. More generally, tensor products of Paulis span all operators on n qubits. A recovery that corrects the relevant Pauli basis errors also corrects their linear combinations, by the quantum error-correction conditions [13]. This is not an assertion that every noise process is literally a classical mixture of Pauli errors.

3.1.1 Shor's Code

Combining protection against bit and phase errors gives Shor's nine-qubit code:

$$\begin{aligned} |0\rangle_L &= \frac{1}{\sqrt{8}}(|000\rangle + |111\rangle)^{\otimes 3}, \\ |1\rangle_L &= \frac{1}{\sqrt{8}}(|000\rangle - |111\rangle)^{\otimes 3}. \end{aligned}$$

The three factors are blocks of three physical qubits. The factor $1/\sqrt{8}$ normalizes each codeword. Checks Z_iZ_j within a block detect a single bit flip in that block. A +1 syndrome does not prove that no error occurred: stabilizers and logical errors can both have trivial syndrome.

For phase-error detection, define

$$R_1 = X_1X_2X_3, \quad R_2 = X_4X_5X_6, \quad R_3 = X_7X_8X_9.$$

Each R_b exchanges $|000\rangle$ and $|111\rangle$ within its block. The checks R_1R_2 and R_1R_3 stabilize both logical states and locate a single block containing a phase error. Errors Z_i and Z_j within the same block act identically on the code, since their product is a stabilizer. These checks, together with the bit-flip checks, allow correction of an arbitrary single-qubit error, including Y .

3.2 Code Concatenation

Concatenation encodes each physical qubit of an outer code using an inner code. Concatenating Shor's $[[9, 1, 3]]$ code with itself gives an $[[81, 1, 9]]$ code. Its distance is at least $3 \cdot 3 = 9$, and a weight-nine nontrivial logical operator attains this bound. Thus an appropriate decoder can correct every error of weight at most four. A simple recursive decoder need not achieve the full distance-based guarantee for every pattern.

Write $|\bar{0}\rangle$ and $|\bar{1}\rangle$ for the normalized inner Shor codewords. There are nine inner blocks, each containing nine physical qubits. The outer logical basis is

$$|\bar{\bar{b}}\rangle = \frac{1}{\sqrt{8}} \left(|\bar{0}\bar{0}\bar{0}\rangle + (-1)^b |\bar{1}\bar{1}\bar{1}\rangle \right)^{\otimes 3}, \quad b \in \{0, 1\}.$$

For example, consider the physical error $E = X_1X_2X_{10}X_{11}$. It acts on the first two qubits of each of the first two inner blocks. Under the usual single-error bit-flip recovery, the inferred corrections are X_3 and X_{12} . The residual operators $X_1X_2X_3$ and $X_{10}X_{11}X_{12}$ act as logical Z on their respective inner Shor blocks. The induced outer error is therefore $\bar{Z}_1\bar{Z}_2$. This is an outer stabilizer, so

$$\bar{Z}_1\bar{Z}_2|\bar{\bar{b}}\rangle = |\bar{\bar{b}}\rangle.$$

In this particular example the logical information is restored after the inner recoveries. This does not mean that every four-qubit bit error is harmless. General decoding algorithms exist, including exhaustive search; the challenge is finding efficient decoders with good performance. Section 3.7 discusses the worst-case complexity of optimal decoding.

3.3 Preliminaries

3.3.1 Some terminology

An $[[n, k, d]]$ quantum code encodes k logical qubits in a 2^k -dimensional subspace of $(\mathbb{C}^2)^{\otimes n}$. For a stabilizer code with $k \geq 1$, its distance is the minimum weight of a Pauli that preserves the code space but acts nontrivially on its logical information. Weight counts nonidentity single-qubit factors; global phases are ignored. In terms of the Pauli centralizer $C(S) = \{P \in \mathcal{P}_n : [P, g] = 0 \text{ for all } g \in S\}$,

$$d = \min\{\text{wt}(P) : P \in C(S), P \notin \langle iI, S \rangle\}.$$

The code corrects arbitrary errors on up to $\lfloor (d-1)/2 \rfloor$ qubits. A code can also correct a more restricted noise model beyond what its distance alone guarantees; the repetition codes above are examples.

3.3.2 Group Theory

A group $(G, \cdot)$ is a set with an operation satisfying the following axioms:

1. $a \cdot b \in G$ for all $a, b \in G$ (closure).
2. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ (associativity).
3. There is $e \in G$ such that $a \cdot e = e \cdot a = a$ for every $a \in G$.
4. Every $a \in G$ has an inverse $a^{-1} \in G$ with $a \cdot a^{-1} = a^{-1} \cdot a = e$.

A subgroup is a subset that is itself a group under the same operation. A set of generators produces every group element by products and inverses; there is no general formula equating the minimum number of generators of an arbitrary finite group with the logarithm of its order.

3.3.3 Pauli Group

The n -qubit Pauli group is

$$\mathcal{P}_n = \{\omega P_1 \otimes \cdots \otimes P_n : P_j \in \{I, X, Y, Z\}, \omega \in \{\pm 1, \pm i\}\}.$$

It has 4^{n+1} elements, including phases. Its elements are unitary, their squares are $\pm I$, and any two either commute or anticommute. Hermitian Paulis have eigenvalues ± 1 and can be used as syndrome observables.

3.3.4 Clifford Groups

The Clifford group is the normalizer of the Pauli group in the unitary group:

$$\text{Cliff}_n = \{U : U \mathcal{P}_n U^\dagger = \mathcal{P}_n\}.$$

Up to global phase, it is generated by single-qubit Hadamard gates, phase gates $S = \text{diag}(1, i)$, and CNOT gates. Normalization means that conjugation maps the Pauli group to itself, not that it fixes every Pauli operator individually.

3.4 An introduction to Stabilizer Codes

3.4.1 Stabilizer Group

A stabilizer group is an abelian subgroup $S \subseteq \mathcal{P}_n$ that does not contain $-I$. Its elements are Hermitian and square to I : an element whose square were $-I$ would violate the assumption. Thus, if S has m independent generators, then

$$S = \langle g_1, \dots, g_m \rangle \cong (\mathbb{Z}_2)^m, \quad |S| = 2^m.$$

Here independence means that no nonempty product of distinct generators is I . Every element has a unique expression $g_1^{a_1} \cdots g_m^{a_m}$ with $a_j \in \mathbb{F}_2$. For example,

$$S = \{III, ZZI, ZIZ, IZZ\} = \langle ZZI, ZIZ \rangle.$$

The corresponding code space is the span of $|000\rangle$ and $|111\rangle$.

3.4.2 Stabilizer States

The stabilizer code space associated with S is

$$V_S = \{|\psi\rangle : g|\psi\rangle = |\psi\rangle \text{ for every } g \in S\}.$$

It is the intersection of the $+1$ eigenspaces of the operators in S , so it is a linear subspace. The condition must hold for $g \in S$, not for every operator in $\mathcal{P}_n$; in particular, $-I$ fixes no nonzero vector.

A *pure stabilizer state* is a normalized state uniquely fixed, up to global phase, by n independent commuting Pauli generators. When $m < n$, V_S is a code space of dimension greater than one; an arbitrary encoded superposition need not itself be a stabilizer state. This distinction matters when applying classical stabilizer-simulation algorithms.

3.4.3 Check matrix

Ignore global phases temporarily and map each single-qubit Pauli to two bits:

$$I \leftrightarrow (0, 0), \quad X \leftrightarrow (1, 0), \quad Z \leftrightarrow (0, 1), \quad Y \leftrightarrow (1, 1).$$

For $S = \langle g_1, \dots, g_m \rangle$, place each generator's X bits and Z bits in a row of $G = (G_X \mid G_Z) \in \mathbb{F}_2^{m \times 2n}$. The j th and $(n+j)$ th entries specify the Pauli acting on qubit j . The signs of the generators must be stored separately: the binary matrix alone cannot distinguish g from $-g$.

For row vectors $u = (x \mid z)$ and $v = (x' \mid z')$, define the symplectic product

$$[u, v]_s = x \cdot z' + z \cdot x' = uJv^T, \quad J = \begin{pmatrix} 0 & I_n \\ I_n & 0 \end{pmatrix}.$$

The corresponding Paulis commute exactly when $[u, v]_s = 0$. Consequently,

$$GJG^T = G_X G_Z^T + G_Z G_X^T = 0$$

is the binary commutation condition. All binary arithmetic here is modulo two.

Proposition. For a valid stabilizer group, its generators are independent if and only if the rows of G are linearly independent over $\mathbb{F}_2$. A dependence would give a product proportional to I ; the stabilizer assumptions rule out $-I$ and $\pm iI$.

Proposition. If $m = n - k$ generators are independent, then $\dim V_S = 2^k$. Indeed, the projector onto V_S is

$$\Pi_S = \prod_{j=1}^m \frac{I + g_j}{2} = \frac{1}{2^m} \sum_{g \in S} g.$$

Only the identity has nonzero trace, so $\dim V_S = \text{tr}(\Pi_S) = 2^{n-m}$.

3.4.4 Unitary dynamics

If $g|\psi\rangle = |\psi\rangle$, then

$$(UgU^\dagger)U|\psi\rangle = Ug|\psi\rangle = U|\psi\rangle.$$

Thus UV_S is fixed by

$$USU^\dagger = \{UgU^\dagger : g \in S\} = \langle Ug_1U^\dagger, \dots, Ug_mU^\dagger \rangle.$$

For a general unitary these conjugated operators need not be Paulis. For a Clifford unitary they remain Paulis, so the binary stabilizer description continues to apply. For instance,

$$HXH^\dagger = Z, \quad HYH^\dagger = -Y, \quad HZH^\dagger = X.$$

Since $|0\rangle$ is stabilized by Z , the state $H|0\rangle = |+\rangle$ is stabilized by X . Updating generators avoids explicitly storing exponentially many amplitudes.

3.4.5 Logical Operators

A nontrivial logical Pauli commutes with every element of S but is not a stabilizer up to global phase. Commutation preserves the code space; multiplication by a stabilizer does not change its action on that space. For k encoded qubits, choose representatives $\bar{X}_i, \bar{Z}_i$ satisfying

$$\bar{X}_i \bar{Z}_j = (-1)^{\delta_{ij}} \bar{Z}_j \bar{X}_i, \quad [\bar{X}_i, \bar{X}_j] = [\bar{Z}_i, \bar{Z}_j] = 0.$$

Two arbitrary nontrivial logical operators need not anticommute; the representatives must be selected in such symplectic pairs.

Computationally, if W is the row space of G , solve

$$GJv^T = 0$$

to obtain the symplectic orthogonal space $W^{\perp_s}$. Then quotient out W and choose a symplectic basis for $W^{\perp_s}/W$, which has dimension $2k$. The ordinary null space of G alone is not the required commutation kernel. Global signs can then be chosen to fix the logical-basis convention.

3.5 Connections between Stabilizer Codes and Classical Linear Codes

The binary space W associated with a stabilizer is linear and satisfies $W \subseteq W^{\perp_s}$. This is the extra commutation constraint absent from an arbitrary classical binary code. If a Pauli error has binary row vector e , its syndrome is

$$s = GJe^T \in \mathbb{F}_2^m,$$

where syndrome bit s_i corresponds to the measured eigenvalue $(-1)^{s_i}$. This resembles a classical parity-check calculation, with the symplectic product replacing the ordinary dot product. The binary weight of $(x \mid z)$ is not generally the Pauli weight: a Y contributes two binary ones but acts on only one qubit.

For a CSS code, generators are either purely X -type or purely Z -type, so

$$G = \begin{pmatrix} H_X & 0 \\ 0 & H_Z \end{pmatrix}, \quad H_X H_Z^T = 0.$$

Classical coding theory can therefore supply suitable check matrices, provided this orthogonality condition is respected. The Steane code is a central example.

3.6 Gottesman–Knill Theorem

Theorem 1 *A circuit starting in a computational-basis product state and using Clifford gates and computational-basis measurements can be sampled efficiently on a classical computer. Efficient classical feed-forward of measurement results also preserves efficient classical simulation.*

Proof sketch. A pure n -qubit stabilizer state is specified by n independent signed generators, requiring $O(n^2)$ bits.

1. A computational-basis input has generators $\pm Z_j$, with signs fixed by its input bits.
2. A Clifford gate U replaces each generator g_j by $Ug_jU^\dagger$. For the elementary gates H , S , and CNOT these updates act on only one or two columns of the tableau, including its sign bits.
3. For a measurement of a Hermitian Pauli P , if P or $-P$ is in the stabilizer, the outcome is deterministic. Otherwise, some generator anticommutes with P , and the outcomes ± 1 are equally likely. Multiply generators to leave just one anticommuting generator, and replace it by the signed measured Pauli. The resulting group describes the post-measurement state.
4. Membership tests, generator products, and all these updates take polynomial time. Sampling and updating each measurement in sequence therefore simulates the full joint outcome distribution, including correlations.

Conjugation updates alone do not prove the theorem for circuits containing measurements; the third step is essential. The result also shows that extensive entanglement by itself does not guarantee an advantage over classical simulation. Many quantum error-correction operations fall inside this simulatable class.

3.7 On the complexity of decoding quantum stabilizer codes

For a specified Pauli noise model, nondegenerate maximum-likelihood decoding chooses the most probable individual error consistent with the syndrome. Suitable decision formulations are NP-complete. In a degenerate code, errors differing by a stabilizer have the same logical effect, so optimal decoding instead maximizes a coset probability:

$$\mathbb{P}(ES \mid s) = \sum_{g \in S} \mathbb{P}(Eg \mid s),$$

with phases ignored. Equal syndrome alone does not imply equal logical effect.

Iyer and Poulin prove that their promise-gap formulation of degenerate maximum-likelihood decoding is #P-complete under polynomial-time Turing reductions, already for $[[n, 1]]$ stabilizer codes on an independent X – Z noise channel [3]. The noise model, promise conditions, and reduction type are part of this statement. Because decoding selects a recovery rather than counts witnesses, the reduction-based formulation matters.

A #P function counts accepting computation paths of a polynomial-time nondeterministic machine. Not every function in #P is hard. The result above is a worst-case hardness statement, not a proof of an unconditional separation from polynomial time or an obstacle to efficient decoders for particular structured code families.

3.8 Importance of nonstabilizer resources for quantum computation

Aaronson and Gottesman’s tableau stores both stabilizer and destabilizer generators, approximately doubling the memory while avoiding Gaussian elimination for deterministic measurements [4]. Their algorithm uses $O(n^2)$ space, $O(n)$ time per elementary Clifford gate, and $O(n^2)$ worst-case time per single-qubit measurement. For g gates and q measurements, a bound including initialization is

$$O(n^2 + gn + qn^2).$$

The improvement concerns measurement updates, not an unconditional $O(n^2)$ total runtime for every stabilizer circuit.

They also show that the decision problem asking whether an output qubit is certainly $|1\rangle$, for their specified nonadaptive stabilizer-circuit model, is $\oplus$ L-complete [4]. The class $\oplus$ L consists of languages accepted by a nondeterministic logarithmic-space machine when the number of accepting paths is odd. This classification does not imply $\oplus$ L $\neq$ P, nor does it apply unchanged to arbitrary classical processing added to the circuit.

Computational-basis input preparation is not a missing ingredient for universal quantum computation: it is already allowed by the theorem. To leave the stabilizer-only setting, one can add a non-Clifford resource, such as a T gate or a suitable magic state. Clifford gates together with T form a universal gate set for approximate quantum

computation [13]. Whether an intermediate circuit model admits an efficient classical simulation depends on its precise allowed gates, inputs, measurements, and classical control; the Gottesman–Knill theorem alone does not classify every such model.

3.9 Examples

In the theoretical discussion, qubits are numbered from one. Python code and circuit diagrams use zero-based indices; the leftmost Pauli letter denotes qubit zero there. All stabilizer signs in the following tables are positive. Logical operators are chosen to match the stated logical basis.

3.9.1 Three-qubit bit-flip code

For $|0\rangle_L = |000\rangle$ and $|1\rangle_L = |111\rangle$, $S = \langle ZZI, ZIZ \rangle$ and one choice of logical representatives is

Element	Operator
S_1	ZZI
S_2	ZIZ
$\bar{X}$	XXX
$\bar{Z}$	ZII

Its full quantum distance is one, because a single Z is a logical error. Nevertheless, it corrects any single X error under the bit-flip noise model.

3.9.2 Three-qubit phase-flip code

Applying $H^{\otimes 3}$ to the preceding code gives $|0\rangle_L = |+++ \rangle$ and $|1\rangle_L = |-- - \rangle$. The stabilizers and logical operators transform by the same conjugation:

Element	Operator
S_1	XXI
S_2	XIX
$\bar{X}$	ZZZ
$\bar{Z}$	XII

This code corrects a single Z error, but also has full quantum distance one. The labels $\bar{X}$ and $\bar{Z}$ depend on the logical basis; swapping them without changing the basis would be inconsistent.

3.9.3 Shor’s Code

For the normalized nine-qubit codewords given earlier, the following generators and logical operators define the $[[9, 1, 3]]$ Shor code:

Element	Operator
S_1	$ZZIIIIII$
S_2	$ZIZIIIII$
S_3	$IIIZZIII$
S_4	$IIIZIZII$
S_5	$IIIIIZZI$
S_6	$IIIIIZIZ$
S_7	$XXXXXXII$
S_8	$XXXIIIXX$
$\bar{X}$	$ZZZZZZZZ$
$\bar{Z}$	$XXXXXXXX$

Multiplication by stabilizers gives equivalent weight-three representatives $\bar{X} = Z_1 Z_4 Z_7$ and $\bar{Z} = X_1 X_2 X_3$. In the chosen basis the former exchanges the logical codewords and the latter gives their relative sign. No nontrivial logical Pauli has weight below three.

3.9.4 Steane Code

The $[[7, 1, 3]]$ Steane code is a CSS construction based on the classical $[7, 4, 3]$ Hamming code C and its $[7, 3, 4]$ dual $C^\perp \subset C$. It is not simply the dual Hamming code regarded as a quantum code. We use the following ordering consistently in both this table and the implementation:

Element	Operator
S_1	XXXXIII
S_2	XXIIXXI
S_3	XIXIXIX
S_4	ZZZZIII
S_5	ZZIIZZI
S_6	ZIZIZIZ
X	XXXXXXXX
Z	ZZZZZZZ

The six generators are independent and commute. The code encodes one logical qubit and corrects every single-qubit error. Reversing the seven physical qubits gives another common convention for its generator table.

3.10 Implementation

STAC supplies stabilizer-code objects, logical operators, and circuit construction routines [12]. To make the example reproducible with the qubit order above, we specify the check matrix explicitly rather than rely on a library's preferred generator ordering. The gate-list algorithms below use NumPy and can also be translated into another circuit library.

3.10.1 Construction of Encoding Circuits for Stabilizer Codes

The construction has three stages: put the check matrix in standard form, choose compatible logical operators, and synthesize an encoding circuit [1]. We first define the Steane code:

```
import numpy as np
import stac

H = np.array([[1, 1, 1, 1, 0, 0, 0],
              [1, 1, 0, 0, 1, 1, 0],
              [1, 0, 1, 0, 1, 0, 1]], dtype=np.uint8)
O = np.zeros_like(H)
G0 = np.block([[H, O], [O, H]])
cd = stac.Code(G0)
stac.print_matrix(G0, augmented=True)
stac.print_paulis(G0)
```

Here $HH^T = 0$ over $\mathbb{F}_2$, and the full matrix is

$$\left(\begin{array}{cccccc|cccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{array} \right).$$

The Pauli output is the six generators in the preceding Steane table. Its rank is six, so $k = 7 - 6 = 1$.

Step 1: Standard form. Elementary row operations change the generator basis without changing the code. Replacing g_i by $g_i g_j$ corresponds to adding binary rows, but Pauli signs must also be tracked. Permuting qubits requires the same permutation in the X and Z halves and a corresponding relabeling of the circuit; it is not an arbitrary independent column operation.

Let $m = n - k$ and $r = \text{rank}(G_X)$ over $\mathbb{F}_2$. Binary row reduction and paired qubit permutations give

$$G = \left(\begin{array}{cc|cc} I_r & A & B & C \\ 0 & 0 & D & E \end{array} \right).$$

The row blocks have sizes r and $m - r$; the column blocks in each half have sizes r and $n - r$. Further reduction of the bottom-right part yields

$$G_{\text{std}} = \left(\begin{array}{ccc|ccc} I_r & A_1 & A_2 & B & C_1 & C_2 \\ 0 & 0 & 0 & D & I_{m-r} & E_2 \end{array} \right),$$

where each half has column-block sizes $r, m - r, k$. For this Steane example, $r = 3$, and one valid standard form is

$$\left(\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{array} \right).$$

It has the same stabilizer group as G_0 and requires no qubit permutation. STAC can obtain a standard form using `cd.construct_standard_form()`. For the explicit calculations below we fix the displayed form as follows:

```
Xrows = np.array([[1, 0, 0, 1, 0, 1, 1],
                  [0, 1, 0, 1, 1, 0, 1],
                  [0, 0, 1, 1, 1, 1, 0]], dtype=np.uint8)
Zrows = np.array([[1, 1, 1, 1, 0, 0, 0],
                  [1, 0, 1, 0, 1, 0, 1],
                  [0, 1, 1, 0, 0, 1, 1]], dtype=np.uint8)
O = np.zeros_like(Xrows)
Gstd = np.block([[Xrows, O], [O, Zrows]])
r = 3
```

Step 2: Logical operators. Compatible binary logical rows are

$$L_X = \left(\begin{array}{ccc|ccc} 0 & E_2^T & I_k & E_2^T C_1^T + C_2^T & 0 & 0 \end{array} \right),$$

$$L_Z = \left(\begin{array}{ccc|ccc} 0 & 0 & 0 & A_2^T & 0 & I_k \end{array} \right).$$

They commute with the stabilizer and have the canonical mutual symplectic products. The additions are modulo two. This helper computes them:

```
def logical_rows(G, r):
    m, twon = G.shape
    n = twon // 2
    k = n - m
    A2 = G[:r, m:n]
    C1 = G[:r, n+r:n+m]
    C2 = G[:r, n+m:]
    E2 = G[r:, n+m:]
    LX = np.hstack((np.zeros((k, r), dtype=np.uint8), E2.T,
                     np.eye(k, dtype=np.uint8),
                     (E2.T @ C1.T + C2.T) % 2,
                     np.zeros((k, n-r), dtype=np.uint8)))
    LZ = np.hstack((np.zeros((k, n), dtype=np.uint8), A2.T,
                     np.zeros((k, m-r), dtype=np.uint8),
                     np.eye(k, dtype=np.uint8)))
    return LX, LZ

LX, LZ = logical_rows(Gstd, r)
```

For this standard form,

$$L_X = (0000111|0000000),$$

$$L_Z = (0000000|1100001).$$

Thus we can use $\bar{X} = IIIIXXX$ and $\bar{Z} = ZZIIIZ$. They are equivalent, up to stabilizers, to the all-X and all-Z representatives in the table.

Step 3: Gate synthesis. Use zero-based qubits $0, \dots, m-1$ as ancillas and put the k input qubits on $m, \dots, n-1$. The following function returns gates in execution order for a valid signed standard form. A sign bit of one denotes minus the tensor product of the indicated Hermitian Paulis. For the Steane example all sign bits are zero.

```
def encoder_ops(G, r, signs=None):
    m, twon = G.shape
    n = twon // 2
    k = n - m
    if signs is None:
        signs = np.zeros(m, dtype=np.uint8)
    LX, _ = logical_rows(G, r)
    ops = []
    for j in range(r, m):          # signs of the pure-Z generators
        if signs[j]:
            ops.append(("X", j))
    for i in range(k):
        for j in range(r, m):
```

```

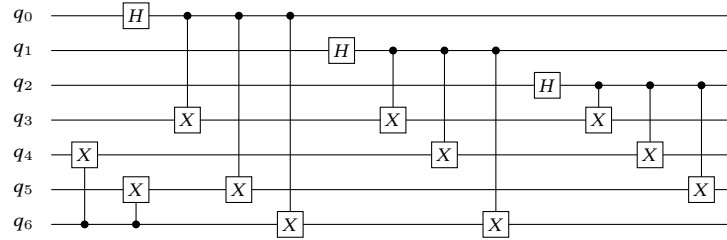
        if LX[i, j]:
            ops.append(("CX", m+i, j))
        controlled = {(1, 0): "CX", (0, 1): "CZ", (1, 1): "CY"}
        for i in range(r):
            ops.append(("H", i))
            if G[i, n+i]: # Y, rather than X, on the pivot
                ops.append(("S", i))
            if signs[i]:
                ops.append(("Z", i))
            for j in range(n):
                if i == j:
                    continue
                x, z = int(G[i, j]), int(G[i, n+j])
                if x or z:
                    ops.append((controlled[x, z], i, j))
        return ops

enc_ops = encoder_ops(Gstd, r)
enc_circ = stac.Circuit.simple(7)
for op in enc_ops:
    enc_circ.append(*op)
enc_circ.draw()

```

Here CY means controlled-Y, with the required phase included. Applying CX and CZ consecutively is not the same gate: for example $CY = (I \otimes S)CX(I \otimes S^\dagger)$, where the rightmost operator acts first. The diagonal-Y and sign corrections in the algorithm are needed for general stabilizers; they are absent in this CSS example.

The encoder is shown below; disjoint gates at the same horizontal position may be applied in either order. The input is on qubit six.



For input $|0\rangle$ it prepares the normalized state

$$\begin{aligned}
 |0\rangle_L = \frac{1}{\sqrt{8}} (&|0000000\rangle + |1100110\rangle + |1111000\rangle + |0011110\rangle \\
 &+ |1010101\rangle + |0110011\rangle + |0101101\rangle + |1001011\rangle).
 \end{aligned}$$

The logical-one state is $|1\rangle_L = \bar{X}|0\rangle_L$, and an arbitrary input is mapped by linearity to $a|0\rangle_L + b|1\rangle_L$. This is encoding, not error correction; no unknown state has been cloned.

3.11 Decoding circuit

For an error-free codeword, applying the inverse encoding unitary gives

$$U_{\text{enc}}^\dagger |\psi\rangle_L = |0\rangle^{\otimes(n-k)} |\psi\rangle.$$

Every gate in the displayed Steane encoder is self-inverse, so reversing its gate order constructs its inverse. For a general encoder, individual gates must also be inverted, for example $S \mapsto S^\dagger$. If noise has acted, recovery is required before this inverse-decoding statement applies.

A second method transfers the logical information to k fresh ancillas without unpreparing the encoded-zero state $|1\rangle$. For one logical qubit, first apply a logical CNOT from the block to an ancilla initialized in $|0\rangle$, then controlled- $\bar{X}$ from the ancilla back to the block:

$$\begin{aligned}
 (a|0\rangle_L + b|1\rangle_L)|0\rangle &\mapsto a|0\rangle_L|0\rangle + b|1\rangle_L|1\rangle \\
 &\mapsto |0\rangle_L(a|0\rangle + b|1\rangle).
 \end{aligned}$$

The first operation is implemented by CNOTs into the ancilla from the support of the pure-Z logical operator $\bar{Z}$. The second is implemented by controlled Pauli factors of $\bar{X}$. These standard-form representatives have positive signs. For multiple logical qubits, repeat the operations using the canonical pairs; they preserve the other logical qubits.

```

def decoder_ops(LX, LZ):
    k, twon = LX.shape
    n = twon // 2
    ops = []
    controlled = {(1, 0): "CX", (0, 1): "CZ", (1, 1): "CY"}
    for i in range(k):
        for j in range(n):
            if LZ[i, n+j]:
                # standard-form LZ is purely Z
                ops.append(("CX", j, n+i))
        for j in range(n):
            x, z = int(LX[i, j]), int(LX[i, n+j])
            if x or z:
                ops.append((controlled[x, z], n+i, j))
    return ops

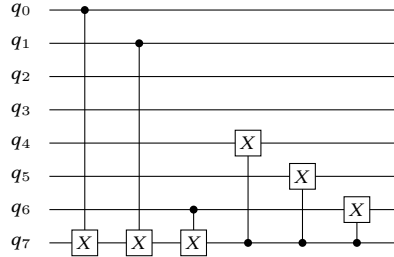
dec_ops = decoder_ops(LX, LZ)
dec_circ = stac.Circuit.simple(8)
for op in dec_ops:
    dec_circ.append(*op)
dec_circ.draw()

```

For the chosen Steane representatives, this gives

$$CX_{0,7}, CX_{1,7}, CX_{6,7}, CX_{7,4}, CX_{7,5}, CX_{7,6},$$

where the first index is the control and the second the target. Its circuit is



A meaningful simulation must supply an encoded input to this circuit. Simulating the decoding gates alone on all-zero physical qubits does not test logical decoding. For input $|0\rangle_L|0\rangle$, this transfer decoder returns $|0\rangle_L|0\rangle$, not eight physical zeros. For a general input it returns $|0\rangle_L|\psi\rangle$, with the recovered qubit on wire seven.

State-vector verification. The following NumPy routine uses the same left-to-right qubit convention as the diagrams. It verifies the exact gate lists without depending on a simulator's display ordering. It covers every gate used in this Steane example.

```

def simulate_steane(state, ops, n):
    state = np.array(state, dtype=complex, copy=True)
    idx = np.arange(2*n)
    for name, *qubits in ops:
        if name == "H":
            q = qubits[0]
            a = np.moveaxis(state.reshape([2]*n), q, 0)
            a = np.stack((a[0]+a[1], a[0]-a[1])) / np.sqrt(2)
            state = np.moveaxis(a, 0, q).reshape(-1)
        elif name == "CX":
            control, target = qubits
            mask = ((idx >> (n-1-control)) & 1).astype(bool)
            permutation = idx.copy()
            permutation[mask] ^= 1 << (n-1-target)
            state = state[permutation]
        else:
            raise ValueError("This example uses only H and CX")
    return state

phi = np.array([np.sqrt(0.3), 1j*np.sqrt(0.7)])
initial = np.zeros(128, dtype=complex)
initial[:2] = phi
# six zero ancillas; phi on q6
encoded = simulate_steane(initial, enc_ops, 7)
recovered = simulate_steane(encoded, enc_ops[::-1], 7)
assert np.allclose(recovered, initial)

zero = np.zeros(128, dtype=complex)
zero[0] = 1
logical_zero = simulate_steane(zero, enc_ops, 7)

```

```
transferred = simulate_steane(np.kron(encoded, [1, 0]), dec_ops, 8)
assert np.allclose(transferred, np.kron(logical_zero, phi))
```

Checking a complex superposition, rather than only logical zero, also tests that relative phase is preserved. The unitary identities above establish the action for arbitrary logical inputs.

3.12 Conclusion

The stabilizer formalism represents quantum codes through commuting Pauli checks and binary symplectic algebra. It makes syndrome measurements, code dimension, logical operators, and Clifford dynamics accessible without storing a full state vector. This efficiency has precise limits: Clifford circuits with stabilizer inputs can be simulated classically, while non-Clifford resources extend the computational model. Optimal decoding poses a different problem, where degeneracy requires accounting for entire logical error cosets. The Steane example illustrates the distinction between encoding, error recovery, inverse decoding, and transfer of logical information to fresh ancillas. Consistent basis conventions and normalization are essential to interpreting these constructions.

References

- [1] Daniel Gottesman (1997). Stabilizer Codes and Quantum Error Correction. Ph.D. thesis, California Institute of Technology. See Chapter 4 for standard form, encoding, and decoding.
- [2] Joschka Roffe (2019). Quantum error correction: an introductory guide. *Contemporary Physics* 60(3), 226–245. doi:10.1080/00107514.2019.1667078.
- [3] Pavithran Iyer and David Poulin (2013). Hardness of decoding quantum stabilizer codes. arXiv:1310.3235. See Theorem 5.1 and the specified promise-gap formulation.
- [4] Scott Aaronson and Daniel Gottesman (2004). Improved simulation of stabilizer circuits. *Physical Review A* 70, 052328. See Sections III and V.
- [5] Lecture notes by Dave Bacon.
- [6] Lecture notes by John Preskill, Chapter 7.
- [7] Andreas Klappenecker, A Short Introduction to Stabilizer Codes.
- [8] Michael E. Cuffaro. On the Significance of the Gottesman–Knill Theorem.
- [9] Arthur Pesah’s blog on quantum computing.
- [10] Exposition by Madhur Tulsiani.
- [11] The Error Correction Zoo.
- [12] Abdullah Khalid. STAC: a Python library for stabilizer codes. API reference.
- [13] Michael A. Nielsen and Isaac L. Chuang (2010). *Quantum Computation and Quantum Information*, 10th anniversary edition. Cambridge University Press.