On the addressability problem on CSS codes

M1 Research Internship at University of Waterloo
Supervised by Samuel Jaques

Jérôme Guyot

March 2024

Abstract

Recent discoveries in asymptotically good quantum codes have intensified research on their application in quantum computation and fault-tolerant operations. This study focuses on the addressability problem within CSS codes, exploring the implementation of gates on individual or subsets of logical qubits. We describe some undesirable codes that we call splitting codes and demonstrate that applying physical Hadamard gates to a subset of physical qubits is feasible only on those splitting codes. There is no such strong condition for phase gates but the codes allowing them are potentially sub-optimal codes. We provide necessary conditions for parallel CNOT gates to be valid logical operators, and we show that some CNOT gate implementations between identical blocks require splitting codes. We also establish conditions for transversal Clifford gates to be valid logical gates and prove that there is no transversal Clifford implementation of $H, HP, PH, CNOT$ on a non-splitting CSS code. Finally, we prove that CSS codes with an asymptotic rate above $\frac{1}{3}$ cannot have addressable logical swaps implemented solely by physical swaps, and that in this same case, we cannot implement all parallel logical CNOTs between two blocks by transversal physical CNOTs involving all physical qubits. This work pioneers the study of addressability in quantum codes, offering new insights and potential directions for future research.

Contents

1 Background 4
1.1 Stabilizer Formalism . . . 4
1.2 CSS codes . . . 6
1.3 Fault tolerant computation . . . 6

2 Valid logical and splitting codes 8
2.1 Single qubit gates . . . 9
2.2 CNOT between two blocks . . . 11
2.3 Transversal Clifford . . . 13

3 Code Automorphisms and Addressability 16

4 Conclusion 18

A Additional background 22
A.1 Quantum computing . . . 22
A.2 Quantum error correcting codes . . . 22
A.2.1 Examples of QECC . . . 22
A.2.2 Basics of error correcting codes . . . 23

B Adressability of P and CNOT gate on a single block 24
B.1 Phase gate . . . 24
B.2 CNOT gate on the same block . . . 25
B.3 Proof for transversal Clifford implementation of PH and HP . . . 26

C Automorphisms 26
C.1 Addressability with fold transversality . . . 27

D Algorithms 27
D.1 Deciding if a code splits . . . 27
D.2 Unitaries implementable by permutations . . . 28

Introduction

Even though quantum computers hold great potential, constructing them in real life is challenging. Classical computers, which use many electrons per bit, are mainly susceptible to bit flip errors. In contrast, quantum computers use a few quantum particles per qubit, which can experience a continuum of possible errors that accumulate, leading to significant errors. Thus, quantum computers are extremely vulnerable to noise and require efficient error correction algorithms to be practical.

Contrary to classical bits, qubits can be subject to two types of errors : bit flips and phase flips. This is the reason why classical error correction does not directly apply to quantum computation. The first quantum error correcting code was Shor's repetition code [Sho95]. This code uses 9 physical qubits to encode a logical qubit, and could protect against any error on one of the 9 physical qubits. Since then better quantum codes have been created such as the toric code by Kitaev [Kit03], or Steane code [Ste96a]. These codes are part of a family of codes called CSS (Calderbank, Shor, Steane) codes. The CSS construction is a way to create a quantum code given two compatible classical codes [CS96][Ste96b], and most well-studied quantum codes fall into this category.

In 2022, Pavel Palateev and Gleb Kalachev created an asymptotically good code [PK22]. This means that as the number of physical qubits n grows, the number of encoded qubits k and the distance d of the code will be linear in n. Such codes were already known for classical computing but the existence of such codes in the quantum context was an open problem. In the same year Anthony Leverrier and Gilles Zemor created quantum tanner codes [LZ22], and proved that they are also asymptotically good. Thus, a lot of research is being done on those codes, and one of the most important question is how can we use those codes for quantum computation : how can we apply gates efficiently on them ?

To perform any computation, operations must be applied to quantum states. One approach is to decode the logical state, apply the necessary operations, and then re-encode it. However, this method is inefficient and leaves the state unprotected while decoded. Ideally, operations should be applied directly to the encoded state. These operations, which might also introduce errors, must be designed to minimize error propagation. For example, in the Shor repetition code, a bit flip error on a physical qubit that is used as a control in a CNOT gate will cause the target qubit to also have a bit flip error. If these qubits are part of the same block (contributing to the same logical qubit), the error becomes uncorrectable. Therefore, in this case, fault-tolerant operations are defined as those that ensure an error in the input leads to at most one error per block in the output, making the error still correctable.

The most common method for fault-tolerant computation is using transversal gates. However, finding transversal implementations of gates is challenging. For CSS codes, it is straightforward to compute valid implementations of logical gates composed of X and Z operations, but these are limited. Thus, the goal is to explore the validity of logical operations involving other gates, which is a complex problem.

Furthermore, now that we are manipulating codes that encode $k > 1$ logical qubits, we want to know if we can implement the logical action of applying a gate U only on the first logical qubit for example, or on a subset of logical qubits. This problem is known as the addressability problem and is equivalent to finding transversal implementation of U_I : the gate U only applied on qubits in I for any unitary U. While a considerable work has been done to find transversal gates and study their actions on the code, very little has been done on the addressability. Addressability is in general mentioned when the gate created is addressable but is rarely a goal : in [Zhu+23] the authors prove that they can implement some addressable logical CZ gates using physical CZ gates on some of the qubits for a family of quasi-hyperbolic color code. As illustrated by this example, such results have only been proved for relatively specific and complex codes.

An easy way to do addressability is to take a code C made of two independent codes C_1, C_2 on which a gate is transversal. However, by doing this, the distance of C is the minimum of the distance of the subcodes. Thus, the goal would be to find codes that have some adressability property, without splitting into independent codes.

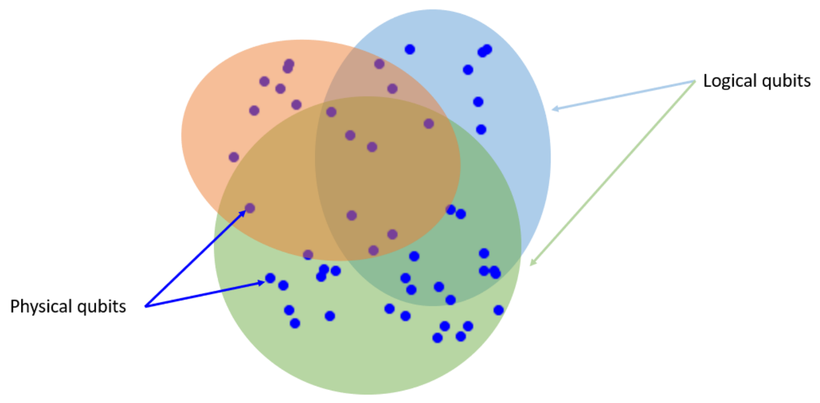


Figure 1: Visualization of a code

In this work, we explore possible implementations of addressable gates on CSS codes. To our knowledge, this is the first study focusing on the addressability problem. We show that implementing addressable logical Hadamard gates using Hadamard gates on a subset of physical qubits is only possible when the code splits. For phase gates, such implementations are possible, but the codes may not be optimal. We describe the necessary conditions for parallel CNOT gates from one block to itself to be a valid logical operator and show that in the case of CNOTs between two identical blocks some implementation require a splitting code. In a more general way, we prove necessary conditions for a transversal Clifford gate to be a valid logical gate and prove in Corollary 2.20 and Theorem 2.21 that there is no transversal Clifford implementation of $H, HP, PH, CNOT$ on a non-splitting CSS code. Finally, by studying the automorphisms of codes, we prove in Corollary 3.11 that any family of CSS code having an asymptotical rate bigger than $\frac{1}{3}$ cannot have addressable logical swaps implemented solely by physical swaps. And in the same case, we cannot implement all addressable logical CNOTs between two blocks by transversal physical CNOTs involving all physical qubits.

1 Background

Let us think of a quantum code as represented in Fig. 1, if we want to apply a logical gate on the orange logical qubit, we will most likely need to apply physical gates on its physical qubits. However, by doing so, we will also modify the structure of the other logical qubits and might break the structure of the code. Hence, the whole problem of this work is to study how one can apply logical gates to some of the logicals qubits without breaking the structure.

1.1 Stabilizer Formalism

We denote by $\mathcal{P}_n$ the group of n-qubits Paulis gates. For example $X \otimes I \otimes X \otimes Z \otimes Y$ would be in $\mathcal{P}_5$. Since all single qubit Pauli gates are their own inverse, we get that it is still true in the n-qubit gates. Hence, the eigenvalues of these operator can be only $+1$ or -1. Furthermore, given any pair of n-qubit Paulis, they either commute or anti-commute.

We call a stabilizer set, any abelian subgroup S of $\mathcal{P}_n$ that does not contain $-I$. Since these operators commute together, they preserve their eigenvalues and are simultaneously diagonalizable. We define $C = \{|\Psi\rangle \ |\forall\ s \in S,\ s\,|\Psi\rangle = |\Psi\rangle\}$ the common $+1$ eigenspace of the stabilizers. Since $-I$ is not in S, C it is not trivial, and it is a vector space : this is a valid linear code. Now if an error e occurred on some codeword $|\Psi\rangle$, we can measure $se\,|\Psi\rangle$ for all $s \in S$ and if for one of them we get $-\,|\Psi\rangle$ then we detect the error. If we do not detect any error, then it mean that the error is a logical operator since it sends codewords to other codewords.

To represent n-qubit Paulis, we often use a representation called the symplectic notation. Any single qubit Pauli can be represented up to sign by a couple $(a, b) \in \mathbb{F}_2^2$ such that (a, b) correspond to $X^a Z^b$. Since any operator is made of n single qubit Paulis, we can represent them up to sign by a vector in $\mathbb{F}_2^{2n}$. when a is a vector we write $X^a = \bigotimes_{a_i=1} X_i$ with X_i being the X gate applied only to qubit i and similarly for Z^a, let s be a stabilizer, we write $s = (s_X \mid s_Z)$ meaning that $s = X^{\otimes s_X} Z^{\otimes s_Z}$. We use the symplectic inner product defined as $\langle s^1, s^2 \rangle_S = \langle s^1_X, s^2_Z \rangle + \langle s^1_Z, s^2_X \rangle$, this means that two n-qubit Paulis commute if and only if their symplectic inner product is even. Once we have a stabilizer group, we can generate this group using $n - k$ generators, and the matrix of the symplectic notation of the generators is a basis of the stabilizers.

We mentioned that an "error" could be detected if and only if it does not send codewords to codewords. We will call "logical operators" those unitaries that preserve the codespace. We call logical identities the unitaries that act as identity on the codespace. Let us show that logical operators are the logical preserving the logical identities.

Proposition 1.1. *L is a logical operator if and only if $LI(C)L^\dagger \subseteq I(C)$ where $I(C)$ is the set of logical identities for the code C.*

Proof. Let $|\Psi\rangle$ be a codeword, then there exists $|\phi\rangle$ in the codespace such that $L|\Psi\rangle = |\phi\rangle$. Let $s \in I(C)$ we get $sL|\Psi\rangle = L|\Psi\rangle$. Hence $LsL^\dagger|\Psi\rangle = |\Psi\rangle$. This means that $LsL^\dagger$ is a logical identity on the codespace, meaning that it is in $I(C)$.

For the other direction, given $LI(C)L^\dagger \subseteq I(C)$, we know that there exists $s' \in I(C)$ such that $sL = Ls'$, which gives $sL|\Psi\rangle = Ls'|\Psi\rangle = L|\Psi\rangle$, hence $sL|\Psi\rangle = L|\Psi\rangle$ for all $s \in I(C)$, meaning that $L|\Psi\rangle$ is in the codespace. □

We call normalizers the element that preserves by conjugation a group G and denote by $N_E(G)$ the normalizer group of a given group G that are in E. Thus the logical operators can be noted $N_E(I(C))$ where E is the space of unitaries, however for simplicity we will just write $N(I(C))$. We can also observe that the stabilizers of a codespace correspond to the restriction of logical identities to Pauli operators. Furthermore, in the context of logicals made of Paulis, there is an easier way to describe valid logicals.

Proposition 1.2. *If $L \in \mathcal{P}_n$ then L is a logical operator if and only if it commutes with the stabilizers.*

Proof. Since L is a logical operator, it sends logical identities to logical identities. And since the Pauli form a group, it sends stabilizers to stabilisers, meaning that $LSL^\dagger \subseteq S$.

Since any two n-qubit Paulis either commute or anti-commute, if there exists $s \in S$ such that $Ls = -sL$ then if L is a logical this gives $LsL^\dagger = -s \in S$ which means that $-I$ is in S, which is absurd since the definition of stabilizers state that $-I \notin S$. □

This allows for an efficient way to compute the Pauli logicals of a given stabilizer code : compute the centralizer of the stabilizer group. And using the symplectic notation it makes things even easier. Given a stabilizer group S, compute a basis of the symplectic representation of S. Compute $S^\perp$ a basis of the space orthogonal to S with respect to the symplectic inner product, this new basis is a basis of the Pauli logicals of S. And since the stabilizer group is abelian, we get that in the symplectic representation, $S \subseteq S^\perp$. As we are in finite dimension, we get that $N(N(S)) = S$ since $(S^\perp)^\perp = S$.

Logical operators can have the same "action" on the code. For example L and Ls with $s \in I(C)$ will have the same action. Hence, a way to consider all the different action is to quotient logical operators by the logical identities (ie. the stabilizers). Hence, $N(I(C))/I(C)$ is the group describing all the different logical actions on the stabilizer code C. We can do something similar by restricting to Paulis and get all the actions of Pauli logical by $N(S)/S$. This makes it possible to easily count the number of logical qubits we have for a given stabilizer code.

The study of non-Pauli logicals is more complex, there is no equivalent of the symplectic notation in the general case which make it harder to analyse. In some cases we can adapt the symplectic notation, for example the XP formalism [WBB22] provides a framework and algorithms to compute all XP logicals : logical gates made of X and $P = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{N}} \end{pmatrix}$ and wI where $w = e^{\frac{2i\pi}{N}}$ and N is an arbitrary precision.

1.2 CSS codes

CSS codes, named after Robert Calderbank, Peter Shor, and Andrew Steane, are a way to create quantum error correcting codes from two classical linear correcting codes C_X, C_Z such that $C_Z^{\perp} \subseteq C_X$. Let us call their parity check matrices H_X, H_Z, the inclusion condition is equivalent to $H_X H_Z^T = 0$. We can describe CSS codes using the stabilizer formalism : let us call X stabilizers the row vectors in the span of H_X and Z stabilizers the row vectors in the span of H_Z. We can see this as saying that the symplectic representation of the stabilizers of the code is $S = \left(\begin{array}{c|c} H_X & 0 \\ 0 & H_Z \end{array}\right)$. We say that H_X corresponds to the X-checks while H_Z correspond to the Z checks. The inclusion condition implies that X and Z stabilizers commute together meaning that we indeed have a well defined stabilizer code. We can see CSS codes as stabilizer codes that can be generated by X only stabilizers and Z only stabilizers.

In this work we will often use another view of stabilizers codes, let us call $A = \text{span}(H_X)$ and $B = \text{span}(H_Z)$, those are the space of the symplectic notations of X and Z checks. In the following, when writing $CSS(A, B)$ we mean $CSS(C_A, C_B)$ with $A = \text{span}(H_X)$ and $B = \text{span}(H_Z)$ with H_X, H_Z the parity checks of C_X, C_Z. This allows us to write that the stabilizers of the CSS code are $S_X = \{X^a \mid a \in A\}$ and $S_Z = \{Z^b \mid b \in B\}$.

The following proposition describes the link between the rate of the CSS code and the classical codes it is made of, where the rate ρ is equal to the ratio $\frac{k}{n} = \frac{n-r}{n}$ where k is the number of logical qubits, r is the dimension of the stabilizer space and n the number of physical qubits.

Proposition 1.3. *Let $C = CSS(C_1, C_2)$, calling ρ', ρ'' the maximum and minimum of the rates of the classical codes C_1, C_2 and ρ the rate of C, we get that $\rho = \rho' + \rho'' - 1$, and $2\rho'' - 1 \leq \rho \leq 2\rho' - 1$.*

Proof. We know that the rate of C is $\frac{n-r}{n}$ where $r = dim(S) = dim(S_x) + dim(S_z)$. Thus we can write $\rho_x = \frac{n-dim(S_x)}{n}$ and $\rho_z = \frac{n-dim(S_z)}{n}$. Hence $\rho = \rho_x + \rho_z - 1$. This directly gives $2\rho'' - 1 \leq \rho \leq 2\rho' - 1$. □

1.3 Fault tolerant computation

Let us assume that we are using a quantum error correcting code that can correct up to t errors. We can then define fault tolerant computation by saying that it should guarantee that if we have less than t errors before the computation, then we should not have more than t after. In a way we are asking for the code not to propagate the error too much. For example, in the case of a code that can correct only a single error (such as Shor's repetition code), assuming there was an X error on qubit 1, then applying a CNOT between qubit 1 and 3 of the code would propagate the error from 1 to 1 and 3, hence making the code not able to correct the error.

We can formalize this idea by introducing f_r the operator that takes a set of words and project it to the set of all those words up to r errors.

Definition 1.4. *Fault Tolerant Gate Error Propagation Property*

Let U be a single qubit gate, it is said to not propagate error if whenever $r + s \leq t$, with s being the number of errors induced by the application of U on the state, we have

$$\boxed{f_r} - \boxed{U} \equiv \boxed{f_r} - \boxed{U} - \boxed{f_{r+s}}$$

Definition 1.5. *Fault Tolerant Gate Correctness Property*

Let U be a single qubit gate, it is said to satisfy the correcness property if whenever $r + s \leq t$, with s being the number of errors induced by the application of U on the state, we have

$$\boxed{f_r} - \boxed{U} - \boxed{\mathcal{D}} \equiv \boxed{f_r} - \boxed{\mathcal{D}} - \boxed{\bar{U}}$$

Where $\mathcal{D}$ represent the ideal decoder and $\bar{U}$ the ideal gate U (ie. without error).

In order to avoid propagation of error, the idea is to isolate some part of the circuit, as no interaction would guarantee that there is less propagation of error, this is the concept on transversality.

Definition 1.6. *Transversality*

Let $\mathcal{Q} = (Q_i)_{i\in I}$ be a partition of the qubits. We say that a gate U is transversal with respect to $\mathcal{Q}$ if it can be decomposed as $U = \bigotimes_{i\in I} U_i$ where U_i acts only on Q_i.

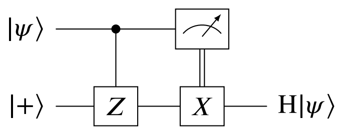
If not mentioned explicitly, the partition is $Q_i = \{i\}$, and for multiple qubit gates involving p blocks of code $Q_i = \{i_j \mid 1 \leq j \leq p\}$ the set of i^{th} qubit of each block.

For example, as we said earlier on the 3 repetition code, one can apply a logical X gate by applying a X gate on all physical qubits. This means that $\bar{X}$, the logical X operator, has a transversal implementation for this code. However the problem of deciding if a logical operation has a transversal implementation is extremely hard and for now (except in some cases, as for example for XP codes when studying the implementability of T gate [WBB22]) the only option is to take an implementation, check that it is a valid logical operator and then study its action.

It is quite easy to see that transversal gates guarantee the absence of propagation of error. However, there are still strong results proving that transversality cannot do everything.

Theorem 1.7. *Eastin-Knill [EK09]*
No quantum error correcting code that can correct single erasure can have universal and transversal gates.

This theorem concerns all quantum error correcting codes and says that we cannot have a set of gates that can be transversal and form a universal set. Thus in order to do fault tolerant quantum computation, transversality cannot be the only solution, it is a very useful tool but need to be paired with a way to go around this theorem in order to obtain a fault tolerant universal set of gates. Since this theorem only apply to unitary implementation, we can use implementation involving measurements to get out of it. Thus the common way to do fault tolerant computation is to take a code having some nice transversal gates, and then generate the gate we need on the other code using state injection. State injection uses an ancilla and link it to a physical qubit, then perform a conditional measurement to implement a gate on this qubit. For example, we can inject the H gate in a code using the following circuit :



However not all gates are as hard to implement fault tolerantly. For example, a lot of codes such as the 15 qubit Reed-Muller codes are built to have transversal T gate with $T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$ as it is less costly to inject an H gate than a T gate. More generally, some codes aim at having a transversal implementation of a gate in the third level of the Clifford hierarchy.

Definition 1.8. *Clifford Hierarchy*
Let $C^1_n = \mathcal{P}_n$ the set of n-qubit Pauli operators.
For $k > 1$, the k^{th} level of the n-qubit Clifford hierarchy is defined as
$C^k_n = \{U \in \mathcal{O}_{2^n}(\mathbb{C}) \mid U\mathcal{P}_n U^\dagger \subseteq C^n_{k-1}\}$.

We call the level 2 of this hierarchy the Clifford group. The operators in this level can be simulated in polynomial time on classical computers [Got98], this result is known as the Gottesman-Knill Theorem. The operators $H, S, CNOT$ all lie in this level and their products can actually generate it. In order to form a universal set of gates and generate any possible unitary, we need to add another gate such as the T gate, that is in the third level of the hierarchy. Using the Eastin-Knill theorem, we get that quantum error correcting codes cannot have transversal $H, CNOT, T$ simultaneously.

2 Valid logical and splitting codes

When we want to implement quantum algorithms such as Shor's algorithm on logical qubits, we need to be able to apply gates to only some of the logical qubits. Hence the need to check what kind of codes allow for those "addressable" gates. In this context, we want to know how we could have adressability. The first method is to apply the gates to only some of the qubits. We show that under such a strategy, it would require the code to split, which we illustrate as a "split" of the basis S of the stabilizers.

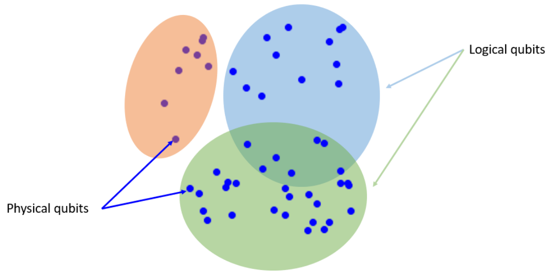


Figure 2: Visualization of a splitting code

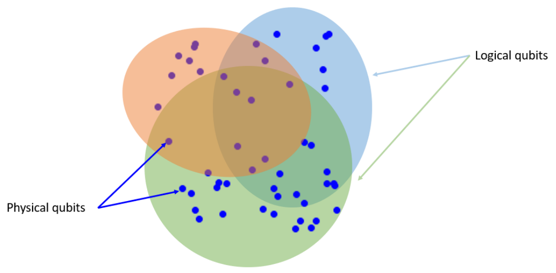


Figure 3: Visualization of a non-splitting code

Definition 2.1. *Let $C = CSS(A, B)$, we say that A splits on some support h if the basis of A can be written up to permutation of the columns as* $\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$ *with h being the support of A_1.*

If A and B both split on some support h, we say that the stabilizer group S splits on h. This is equivalent to saying that C splits into two independent codes C_1, C_2 where C_1 is made of the qubits in h and C_2 of the rest of the qubits. In this case C_1, C_2 are both CSS codes.

Remark 2.2. *The notion of independence comes from the separable nature of the system.*

If, using the previous definition, a CSS code C splits, we can observe that C is equivalent to two independent codes C_1, C_2. Hence, if some gate U has a transversal implementation on C_1 and C_2, it trivially has an addressable equivalent in C : we can chose to which code we apply U on. This case is not interesting and it also means that the distance of the code is smaller than the distance of both subcodes C_1, C_2. Importantly, this means that the distance of C can be at most the maximum of the distance of the two subcodes. If we start with a high-rate code (like those mentioned in the introduction), we can expect them not to split because otherwise there would be an even smaller, better construction with equally good error correction capabilities. Moreoever, we could apply the same argument to these subcodes, and eventually the codes must stop splitting because we know that the rate of a single-qubit code cannot be arbitrarily high. In short, we can regard a code that splits as a pathological case and ignore it. However, we will show in this section that in many cases, it is impossible to apply gates addressably unless the code splits. Specifically, the following addressable gates are impossible for non-splitting codes:

- Logical H, HP, PH gates made of physical single-qubit Cliffords (Theorem 2.19)

- Logical CNOT gates made out of physical CNOTs, where the control and targets share a specific overlap (Theorem 2.12)
- Logical CNOT gates made of physical single-qubit Cliffords (Theorem 2.21)
- Logical CNOT gates made out of physical CNOTs, where the logical target has weight greater than the logical control (Proposition 2.9)

In order to prove these results, we will use another description of splitting codes proved in the following lemma.

Lemma 2.3. *A splits on some support $h \Leftrightarrow \forall\ a \in A,\ a \cap h \in A$.*

Proof. If A splits on some support h then the matrix of A can be written up to permutation of qubits as $\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$ with A_1 having h as support. Thus $A = (A_1\ 0) \oplus (0\ A_2)$. Hence for all $a \in A$, $a = (a_1\ a_2)$ and $(a_1\ 0) = a \cap h \in A$ as $A = (A_1\ 0) \oplus (0\ A_2)$.

Now for the other direction, if $\forall\ a \in A,\ a \cap h \in A$, then considering $(a_i)_i$ a basis of A, we get that $(a_i \cap h)_i$ is a generating set of the restriction of A to h. We can take a basis from it, and since $a_i \cap h \in A$ by hypothesis, we are able to get the top left block of the matrix : the basis of $(A_1\ 0$.

Now since we started with a basis of A, we can also generate the part outside of h using them. We consider the generating set $(a_i \backslash h)_i$ and make it into a basis. Since $a_i + a_i \cap h = a_i \backslash h$ we get that $a_i \backslash h \in A$, we indeed have a valid basis of A of the form $\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$ which by definition means that A splits on h. □

Remark 2.4. *We will use the abusive notation $A \cap h \subseteq A$ to mean $\forall\ a \in A,\ a \cap h \in A$.*

Remark 2.5. *We describe in Appendix D.1 an algorithm to decide if a code split and in the case of a split, return the different blocks.*

2.1 Single qubit gates

This first implementation we can think of is to apply physical gates to some carefully chosen physical qubits, for example the qubits that are part of the logicals qubits we want to modify.

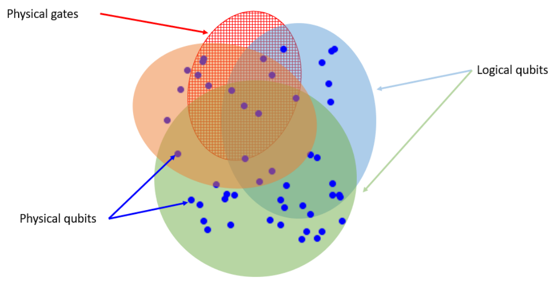


Figure 4: Visualization of physical gates

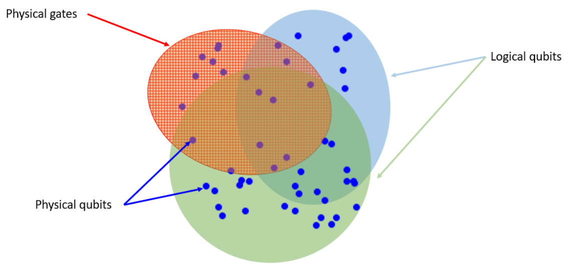


Figure 5: Visualization of an example of a naive implementation

Proposition 2.6. *Let $C = CSS(A, B)$ with stabilizers S, if H^h is a valid logical then S splits on h. And if $A = B$, then S splits on some support h if and only if H^h is a valid logical.*

Proof. For $\bar{H} = H^h$ to be a valid logical we need $\bar{H}S\bar{H} \subseteq S$. Let $s \in S_x$, $s = X^a$, then $\bar{H}s\bar{H} = Z^{h\cap a}X^{a\backslash h}$. Thus $\bar{H}s\bar{H} \in S \Leftrightarrow Z^{h\cap a}, X^{a\backslash h} \in S$. And as X^a is a stabilizer, we get that $X^aX^{a\backslash h} = X^{a\cap h} \in S$. By doing the same with a Z stabilizer we get the following properties :

$\forall\, a \in A,\ a \cap h \in A$ and $\forall\, b \in B,\ b \cap h \in B$. Since all stabilizers in a CSS code can be generated by all-Z or all-X this characterization is enough. Hence if $\bar{H}$ is a valid logical, then it means that S splits on the support of h.

Now in the case of $A = B$ we do the other implication. If S splits then $N(S) = S^{\perp}$ splits, hence $S^{\perp} = S_1^{\perp} \oplus S_2^{\perp}$. Furthermore, since $A = B$, this property of same support for X and Z is also kept for the logicals. Hence, if X^c is a valid logical, then Z^c is also a logical. Now applying H^h where h corresponds to the support of $S_1^{\perp}$ or $S_2^{\perp}$ gives us a valid logical that sends X^c to Z^c. Furthermore, calling $\bar{X}_i, \bar{Z}_i$ the logical X, Z on logical qubit i, if $\bar{X}_i = X^c$ and $\bar{Z}_i = Z^c$ for all logical qubits i in the support of h, then H^h implements a valid logical H gate on all the qubits encoded on his support.

□

This means that if H^h is a valid logical, the code has to split into two codes made respectively of the qubits from the support of h and those that are not in the support of h. Furthermore, for addressability, we need such a property to hold as many times as we have logical qubits in order to be able to apply the gate to only one of the logical qubits.

Corollary 2.7. *Let $C = CSS(A, B)$ with stabilizers S, if C admits addressable gates implemented by physical H gates then S splits into k blocks.*

Proof. Let us assume that S admits an addressable logical gate U made of physical H. It means that we have k independent logicals $\bar{U}_i = H^{h_i}$. Using Proposition 2.6 we get that S will splits into k blocks formed by the supports of the h_is. □

These results show that in the context of CSS codes, applying physical Hs to blocks of qubits is a good strategy only if the code splits. We can use the same strategy and obtain inclusions depending on A and B for the other gates, the results are proved in appendix and summarized in the following table.

Gate	CSS(A,B)	if $A = B$
H^h	$A \cap h \subseteq A$ $B \cap h \subseteq B$	$A \cap h \subseteq A$
P^h	$A \cap h \subseteq B$	$A \cap h \subseteq A$
$CNOT^{I \to J}$ in the same block	$\pi_R(A \cap I) \subseteq A$ $\pi_R(B \cap J) \subseteq B$	$A \cap I \subseteq A$ $A \cap J \subseteq A$

Table 1: Inclusions if the gates are valid logicals

Where the CNOTs are applied on one block of the code, and the π_R function is the bijective function between target and control qubits given by the CNOTs.

We can also get some positive results for the P gate. The following corollary is built on Proposition B.1 studying the addressability of P and state that when the code satisfies some hypothesis we can build a non trivial logical with phase gates on a subset of qubits.

Corollary 2.8. *Let $C = CSS(A, B)$ such that there exists some $h = Supp(B_1)$ subset of $[\![n]\!]$ satisfying $\forall\ a \in A, a \cap h \in B$ and $|a \cap h| = 0 \bmod 4$ then P^h is a valid logical, and if there is a non trivial X logical with support in h, then P^h is a non trivial logical.*

Proof. Using the proof of Proposition B.1 we get that in this case P^h is a valid logical.
Now about the action, assuming it exists, let us take X^c a non trivial X logical with support contained in h. We get that $c \in B^{\perp}$ and since $A \cap h \subseteq B$, we have that $B^{\perp} \subseteq (A \cap h)^{\perp}$. Thus $c \in (A \cap h)^{\perp}$, and this means that $c \in A^{\perp}$ as c is zero outside of h. Thus Z^c is a valid logical in

this code. Now as $c \notin B$ since the logical is non trivial and $A \cap h \subseteq B$, we get that $c \notin A \cap h$, thus $c \notin A$. Hence $c \in A^{\perp} \backslash A$, which means that Z^c is a non trivial logical.

Since P^h sends non trivial logical X^c to $i^c X^c Z^c \not\equiv X^c$ as Z^c is a non trivial logical. We get that P^h is a non trivial logical operator on the code.

□

Furthermore, in Appendix B.1 we describe a way to build such codes allowing addressable phase gate. However, we then show that these codes are more likely to have small distances Appendix B.1. It is not obvious how one would build codes having addressable phase gate and a good distance.

We can now use those equations, for example on the CNOT to study the possibility of implementing addressable CNOT using physical CNOTs. For this, let us introduce $\tilde{X}_i = X^{\tilde{a}_i}$ be such that $|\tilde{a}_i| = \min\{|a| \mid X^a \equiv \bar{X}_i\}$ where $\bar{X}_i$ is the logical X on logical qubit i. Thus $\tilde{X}_i$ is the minimal representative of $\bar{X}_i$. We can define similarly $\tilde{Z}_i = Z^{\tilde{b}_i}$.

Let us now show that using physical CNOTs to implement addressable logical CNOTs has some restriction.

Proposition 2.9. *Let us assume that physical $CNOT_{I \to J}$ implements logical $CNOT_{I' \to J'}$. Then for all logical qubit i' sent on j' in the logical CNOT, we have $|\tilde{a}'_i| \geq |\tilde{b}'_j|$.*

Proof. Using the equations for CNOTs on one block, we get that $\pi(\tilde{a}'_i \cap I) \equiv \bar{Z}'_j$.
Hence $|\tilde{a}'_i| \geq |\tilde{a}'_i \cap I| \geq |\tilde{b}'_j|$. □

2.2 CNOT between two blocks

Let us now consider the case where we take CNOTs between two blocks. Compared to the one block case, we can allow I, J (the set of control qubits and set of target qubits) to share elements, meaning that calling R the bijective relation such that iRj iff $CNOT(i,j)$ is applied, $R \subseteq [\![n]\!]^2$.

We now have to define two permutations :
Let π_I be defined such that

$$\begin{cases} \text{If } i \in I, \pi_I(i) = j \text{ such that } (i,j) \in R \\ \text{If } j \in J \backslash I, \pi_I(j) = i \in I \backslash J \\ \text{Else } x \notin I \cup J, \pi_I(x) = x \end{cases}$$

We define similarly π_J

$$\begin{cases} \text{If } j \in J, \pi_J(j) = i \text{ such that } (i,j) \in R \\ \text{If } i \in I \backslash J, \pi_J(i) = j \in J \backslash I \\ \text{Else } x \notin I \cup J, \pi_J(x) = x \end{cases}$$

Since there are as many controls not being targets and targets not being controls, π_I and π_J are well defined. We can take $\pi_J = \pi_I^{-1}$ for simplicity.
We also define f_I which is similar to π_I but without being a permutation, and same for f_J:

$$\begin{cases} \text{If } i \in I, f_I(i) = j \text{ such that } (i,j) \in R \\ \text{Else } x \notin I, f_I(x) = x \end{cases}$$

Using the method from Proposition B.4, we obtain similar equations for the case of two blocks :

$$\begin{cases} \forall\, a \in A, \pi_I(a \cap I) \in A' \\ \forall\, b' \in B', \pi_J(b \cap J) \in B \end{cases} \tag{1}$$

where we did the CNOTs with control on code C and target on code C'.

Proposition 2.10. *In the case of $A = B = A' = B'$ we get that S must split on I and J.*

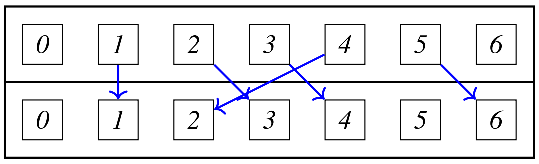
Proof. The technique is the same as for Corollary B.5, we only have to observe that $\pi_J(\pi_I(a \cap I)) = a \cap I$ and $\pi_I(\pi_J(b' \cap J)) = b' \cap J$. □

Let us now apply the CNOTs between two blocks of the same code, meaning that $A = A'$ and $B = B'$. We show that under certain hypothesis on the orbits of π_I, then the code has to split. To illustrate this, let us start by the study of an example.

Example 2.11. *Let us consider on a code with 7 qubits, and the following unitary*

$$CNOT_{I \to J} = CNOT(1,1) \otimes CNOT(1,1) \otimes CNOT(2,3) \otimes CNOT(3,4) \otimes CNOT(4,2) \otimes CNOT(5,6)$$

The CNOTs can be represented using the following diagram, where the top qubits are from the control block and the bottom ones are the targets.



We can easily see that $I = \{1,2,3,4,5\}$ and $J = \{1,2,3,4,6\}$

Furthermore, the only way to define π_I and π_J in this case is $\pi_I = (0)(1)(2,3,4)(5,6)$ and $\pi_J = \pi_I^{-1}$

By observing π_I we can see that there are two orbits contained in I : (1) and (2,3,4)

Meaning that using Theorem 2.12, the code splits on $\{1,2,3,4\}$. But let us develop the proof on this example to understand better. Let us assume that this unitary is a valid logical, and assume that $a = 1101001$ is an element of A. Then since we have a valid logical, we get that $\pi_I(a \cap I) \in A$, we have $a \cap I = 0101000$ and $\pi_I(a \cap I) = 0100100$ is in A. Now this does not have the form of $a \cap \bar{I}$, so as in the proof we can apply this reasoning one more time. Let us call $a_1 = \pi_I(a \cap I) = 0100100$. We now compute $a_2 = \pi_I(a_1 \cap I) = 0110000$ and $a_3 = \pi_I(a_2 \cap I) = 0101000$. We know that $a_3 \in A$ and we can observe that $a_3 = a \cap \{1,2,3,4\}$ where $\{1,2,3,4\}$ is the support of the orbits of π_I contained in I. We can repeat this process for all X stabilizer and use π_J for the Z stabilizer and obtain the split of the code.

By formalizing what we developed in this example, we can actually show that if such a unitary is a valid logical, then the code has to split on the support of some of its orbits.

Theorem 2.12. *Let C be a CSS code such that $S = \{X^a, Z^b \mid a \in A, b \in B\}$, with $A, B \subseteq \mathbb{Z}_2^n$. If $CNOT^{(i,j)_R}$ is a valid logical then S splits on the union of the support of the orbits of π_I contained in I.*

Proof. Since π_I and π_J are bijective, we have the equality $\pi_I(a \cap I) = \pi_I(a) \cap \pi_I(I)$, and similarly for π_J. Hence iterating this equation N times gives

$$\pi_I(\pi_I(\cdots \pi_I(a \cap I) \cap I) \cdots) \cap I) = \pi_I(a)^N \cap \left(\bigcap_{1 \le k \le N} \pi_I^k(I) \right)$$

Which means that

$$\forall\, a \in A, N \in \mathbb{N}_{\rangle 0},\ \pi_I(a)^N \cap \left(\bigcap_{1 \le k \le N} \pi_I^k(I) \right) \in A$$

$$\forall\, b \in B, N \in \mathbb{N}_{\rangle 0},\ \pi_J(b)^N \cap \left(\bigcap_{1 \le k \le N} \pi_J^k(J) \right) \in B$$

However, since π_I is a permutation, there exists some order N_0 such that $\pi_I^{N_0} = Id$. Thus by calling $\bar{I} = \bigcap_{1 \le k \le N_0} \pi_I^k(I)$, we get that A splits on $\bar{I}$ and a similar results holds for B with $\bar{J}$ defined by a similar way with π_J. For this splits to be interesting, it must be non-trivial, meaning we want $\bar{I}, \bar{J}$ not to be empty.

We can observe that since π_I and f_I are defined similarly on I we get that

$$f_I(f_I(\cdots f_I(I) \cap I) \cap I) \cdots) \cap I) = \pi_I(\pi_I(\cdots \pi_I(I) \cap I) \cap I) \cdots) \cap I)$$

Furthermore, since for all $x \notin I$, $f_I(x) = x$, we get that $f_I(f_I(I) \cap I) \cap I = f_I(f_I(I)) \cap I$. First the left inclusion is obvious since $f_I(I) \cap I \subseteq f_I(I)$. Now for the other direction, given $y \in f_I(f_I(I)) \cap I$ then as $y \in I$, then the element z such that $f_I(f_I(z)) = y$ must verify $f_I(z) \in I$, otherwise, since f is the identity outside of I, we get that $y = f_I(z) \notin I$ which is absurd. Thus, by iterating this identity N times we get :

$$f_I(f_I(\cdots f_I(I) \cap I) \cap I) \cdots) \cap I) = f_I^N(I) \cap I$$

Thus, we can define $\bar{I}$ as $\bar{I} = f_I^{N_0}(I)$ and $\bar{J} = f_J^{N_0}(J)$, where N_0 is the same since we can take π_I, π_I to be inverse of each other.

Now, if π_I contain at least an orbit having support in I then $\bar{I}$ is non-empty. Indeed, let us call C_I the support of this orbit, for all k, $C_I \subseteq \pi_I^k(I)$, thus $C_I \subseteq \bar{I}$. Hence, if C_I is non-empty, then $\bar{I}$ is non-empty.

In the case of π_I not having any orbit in I, we can use that f_I would not have one either since they are defined the same on I. This means that for all $i \in I$ there exists n_i such that $f_I^{n_i}(i) \notin I$. Hence, by taking $n = \max_i n_i$, we have that $f_I^n(I) \cap I = \emptyset$, and $\bar{I} = \emptyset$.

Let us now consider a CNOT such that is has an orbit in I, then by definition, this orbit is in $I \cap J$ since each of its element is both a target and a control. And as we can take π_J to be the inverse of π_I, if we have an orbit in I for π_I, then we have an orbit for π_I, π_J in $I \cap J$. Thus, $\bar{I} = \bar{J} = \bigcup_i |C_i|$ where $(C_i)_i$ are the orbits of π_I on I. Furthermore, using the equations above we get that A, B split on the same support $\bar{I}$, meaning that S splits on $\bar{I} = \bar{J}$.

□

2.3 Transversal Clifford

Before studying the entire Clifford group, we study the case of unitaries made of a tensor product of n single qubit Clifford gates. This means that we consider $U = \bigotimes_{i \leq n} U_i$ where $U_i \in \langle iI, X, Z, P, H \rangle$. To make it easier, we classify those Clifford gates depending on their action on the Pauli gates. Let us write $\mathcal{C}/\mathcal{P} = \langle H, P \rangle$ be the single qubit Clifford quotiented by the Paulis. Let us consider this up to global phase, we then have 6 different type of gates:

- type I : sends X to X and Z to Z
- type P : sends X to Y and Z to Z
- type HP : sends X to Y and Z to X
- type PHP : sends X to X and Z to Y
- type PH : sends X to Z and Z to Y
- type H : sends X to Z and Z to X

Now since we might want to send on $-X$ or $-Z$ etc, we can add the Paulis, For example to put a minus on the image of X we can put a Z, for the image of Z, take an X and for both we can take iY. Hence, any single qubit Clifford gate can be written $U_i = AB$ with $A \in \langle I, H, P \rangle$ and $B \in \langle I, X, Z \rangle$.

Let us first consider the unitaries up to phase, meaning that we only care of the unitaries quotiented by the Paulis. Let U be a unitary made of a tensor product of single qubit Clifford quotiented by Paulis such that $U_i \in \langle H, P \rangle$. We write $\{U^G\} = \{i \mid U_i \text{ is of type } G\}$. For example, if we take $U = P \otimes H \otimes H$, we have $\{U^H\} = \{2, 3\}$. And by $\{U^{G_1, G_2}\}$ we denote $\{U^{G_1}\} \cup \{U^{G_2}\}$.

Proposition 2.13. *Let C be a CSS code such that $S = \{X^a, Z^b \mid a \in A, b \in B\}$, with $A, B \subseteq \mathbb{Z}_2^n$. Let U be a Clifford gate made of a tensor product of n single qubit Clifford gate, and $\bar{U}$ the unitary U quotiented by Paulis. Then if U is a valid logical, we have that*

$$\begin{cases} \forall\, a \in A, a \cap \{\bar{U}^{P,HP,PH,H}\} \in B \\ \forall\, a \in A, a \cap \{\bar{U}^{PH,H}\} \in A \\ \forall\, b \in B, b \cap \{\bar{U}^{HP,PHP,PH,H}\} \in A \\ \forall\, b \in B, b \cap \{\bar{U}^{HP,H}\} \in B \end{cases} \tag{2}$$

Proof. Using the types of single qubit Clifford we get the action of $\bar{U}$ on X^a and Z^b :

$$\begin{aligned} \bar{U} X^a \bar{U} &= X^{a\setminus\{\bar{U}\}} X^{a\cap\{\bar{U}^{I,PHP}\}} Z^{a\cap\{\bar{U}^{PH,H}\}} Y^{a\cap\{\bar{U}^{P,HP}\}} \\ &= X^{a\setminus\{\bar{U}\}} X^{a\cap\{\bar{U}^{I,P,HP,PHP}\}} Z^{a\cap\{\bar{U}^{P,HP,PH,H}\}} \\ &\equiv X^{a\cap\{\bar{U}^{PH,H}\}} Z^{a\cap\{\bar{U}^{P,HP,PH,H}\}} \quad \text{as } X^a \text{ is a stabilizer} \end{aligned}$$

We can use the same method on the Z stabilizers to obtain the two other equations.

□

Furthermore, we can combine those equations and get some more splitting :

Corollary 2.14. *Let $C = CSS(A, B)$, and U be a Clifford gate made of a tensor product of n single qubit Clifford gate, and $\bar{U}$ the unitary U quotiented by Paulis. Then if U is a valid logical, we have that*

$$\begin{cases} \forall\, s \in S, s \cap \{\bar{U}^{HP}\} \in S \\ \forall\, s \in S, s \cap \{\bar{U}^{PH}\} \in S \\ \forall\, s \in S, s \cap \{\bar{U}^{H}\} \in S \end{cases} \tag{3}$$

Proof. Let $a \in A$ then using the second equation, $a \cap \{\bar{U}^{PH,H}\} \in A$, and using the first equation, we get that $a \cap \{\bar{U}^{PH,H}\} \in B$. Now using the fourth equation, $(a \cap \{\bar{U}^{PH,H}\}) \cap \{\bar{U}^{HP,H}\} = a \cap \{\bar{U}^{H}\} \in B$. Finally, using the third equation, we get that $a \cap \{\bar{U}^{H}\} \in A$. We can do something similar for B and show that S slits on $\{\bar{U}^{H}\}$.

Now that we have those splits, we can combine them with the second and fourth equations of Proposition 2.13 to obtain that

$$\forall\, a \in A, a \cap \{\bar{U}^{PH}\} \in A \text{ and } \forall\, b \in B, b \cap \{\bar{U}^{HP}\} \in B$$

We can even go further, let us take $a \in A$, then using the first equation of Proposition 2.13 we get $a \cap \{\bar{U}^{P,HP,PH,H}\} \in B$, now using the newly found $b \cap \{\bar{U}^{HP}\} \in B$, we get that $a \cap \{\bar{U}^{HP}\} \in B$. Finally, by using that $b \cap \{\bar{U}^{HP,PHP,PH,H}\} \in A$ we get that $a \cap \{\bar{U}^{HP}\} \in A$. This implies that the stabilizers split on $\{\bar{U}^{HP}\}$. Now by using the same analysis on B with $\{\bar{U}^{PH}\}$ we can show that B splits on $\{\bar{U}^{PH}\}$ and thus the stabilizers split on $\{\bar{U}^{PH}\}$. □

Remark 2.15. *The implication of Corollary 2.14 is quite important, because if C is not a splitting code then $\{\bar{U}^{HP}\}, \{\bar{U}^{PH}\}, \{\bar{U}^{H}\}$ are all either empty or the whole set of qubits.*

We can apply those results to see how one could build a logical Hadamard made with a transversal Clifford. For each logical operator $\bar{X}_i, \bar{Z}_i$ we can take as representative any gate that is in $\bar{X}_i S$ since they have the same action as $\bar{X}_i$. Let us then define as $\tilde{X}_i = X^{\tilde{a}_i}$ the element of $\bar{X}_i S_X$ with the smallest Hamming weight and similarly for $\tilde{Z}_i = Z^{\tilde{b}_i}$.

Proposition 2.16. *Let U be a Clifford gate made of a tensor product of n single qubit Clifford gate, and $\bar{U}$ the unitary U quotiented by Paulis. Then if U sends $\bar{X}_i S$ to $\bar{Z}_i S$ and conversely, we have that $\tilde{a}_i \subseteq \{\bar{U}^{PH,H}\}$ and $\tilde{b}_i \subseteq \{\bar{U}^{HP,H}\}$.*

Proof. Using the same analysis as in the previous proof, $\bar{U} X^{\tilde{a}_i} \bar{U} = X^{\tilde{a}_i\setminus\{\bar{U}\}} X^{\tilde{a}_i\cap\{\bar{U}^{I,P,HP,PHP}\}} Z^{\tilde{a}_i\cap\{\bar{U}^{P,HP,PH,H}\}}$. However, since we are sending $\bar{X}_i S$ to $\bar{Z}_i S$, it means that $X^{\tilde{a}_i\setminus\{\bar{U}\}} X^{\tilde{a}_i\cap\{\bar{U}^{I,P,HP,PHP}\}}$ is a stabilizer. However if it was the case, then we could consider $a = \tilde{a}_i + \tilde{a}_i\setminus\{\bar{U}\} + \tilde{a}_i \cap \{\bar{U}^{I,P,HP,PHP}\}$ and it would have a smaller Hamming weight. Hence the minimality of the Hamming weight of $\tilde{a}_i$ guarantees that $\tilde{a}_i \backslash \{\bar{U}\} + \tilde{a}_i \cap \{\bar{U}^{I,P,HP,PHP}\} = 0$. Hence $\tilde{a}_i \subseteq \{\bar{U}^{PH,H}\}$. Using the exact same method on $Z^{\tilde{b}_i}$ we get that $\tilde{b}_i \subseteq \{\bar{U}^{HP,H}\}$.

□

Remark 2.17. *Using the same method, if we want U to act as identity on $\bar{X}_i$ and $\bar{Z}_i$ then it implies $\tilde{a}_i \cap \{\bar{U}^{PH,H}\} = 0$ and $\tilde{b}_i \cap \{\bar{U}^{HP,H}\} = 0$.*

Remark 2.18. *In the case where we want U to act as an H on all logical qubits simultaneously, calling $\tilde{A} = \bigcup_{i \leq k} \tilde{a}_i$ and $\tilde{B} = \bigcup_{i \leq k} \tilde{b}_i$ we get that for U to be valid we need $\tilde{A} \cap \tilde{B} \subseteq \{\bar{U}^H\}$, meaning that $\bar{U} = H^{\tilde{A} \cap \tilde{B}} \otimes \bar{U}'$.*

Theorem 2.19. *On a CSS code, any implementation of addressable logical Hadamard using transversal Clifford requires a splitting code. Furthermore, the logical Hadamard is applied to all logical qubits of a subcode and implemented using physical Hadamard on all qubits of this subcode.*

This result stays valid when replacing $\bar{H}_I$ the logical hadamard on qubits in I by logical HP_I or logical PH_I.

Proof. Let $C = CSS(A, B)$ be a CSS code. Let U be a tensor product of n single qubit Clifford gates. Using Corollary 2.14 we get that for U to be a valid logical gate, we need

$$\begin{cases} \forall\, s \in S, s \cap \{\bar{U}^{HP}\} \in S \\ \forall\, s \in S, s \cap \{\bar{U}^{PH}\} \in S \\ \forall\, s \in S, s \cap \{\bar{U}^{H}\} \in S \end{cases}$$

Now let us consider X_i a logical operator on this code. Since the Z stabilizers split, we can take X_i with support either in $\{\bar{U}^{HP}\}$ or $\{\bar{U}^{H}\}$ or $\{\bar{U}^{PH}\}$ or $\{\bar{U}^{I,P,PHP}\}$, and its minimal representative $\tilde{X}_i$ will have support in the same block. We can apply the same analysis for $\tilde{Z}_i$. Now using Proposition 2.16, we know that in order to act as a logical Hadamard on logical qubit i we need $\tilde{a}_i \subseteq \{\bar{U}^{PH,H}\}$ and $\tilde{b}_i \subseteq \{\bar{U}^{HP,H}\}$. However, since they have to anti commute, their intersection cannot be empty. Hence, they both have to be contained in the block $\{\bar{U}^H\}$.

Now for logical qubits we do not want to modify, using Remark 2.17 we need $\tilde{X}_i$ to have its support outside of $\{\bar{U}^{PH,H}\}$ and $\tilde{Z}_i$ outside of $\{\bar{U}^{HP,H}\}$. Calling C' the code corresponding to the qubits in $\{\bar{U}^{I,P,HP,PHP,PH}\}$ and calling C_H the code with qubits in $\{\bar{U}^H\}$, we can write $U = U' \otimes H^h$ with $h = \{\bar{U}^H\}$. Finally, for U to act as a logical addressable Hadamard, we need U' to be a logical identity in C' and C_H to be a self-dual code ($A \cap \{\bar{U}^H\} = B \cap \{\bar{U}^H\}$). This way we are going to apply a logical Hadamard to all logical qubits of C_H.

The proof in the case of logical HP_I or logical PH_I is similar and has been done in Theorem B.7.

□

Corollary 2.20. *There is no transversal Clifford implementation of addressable logical H, HP, PH on a non-splitting CSS code.*

Theorem 2.21. *There is no transversal Clifford implementation of addressable logical CNOT on a non-splitting CSS code.*

Proof. Suppose the logical CNOT acts from a logical qubit set I to a target set J. Let X^a be equivalent to $\bar{X}_I$: the logical gate with $\bar{X}$ on all the control logical qubits. Using Corollary 2.14, we know that we can only use gates of type I, P, PHP in the transversal Clifford implementation. Let us call U this implementation. We have $UX^aU^\dagger = X^a Z^{a \cap \{\bar{U}^P\}}$. Since we want to act as a logical CNOT, this is supposed to be equivalent to X^{a+b}, where X^b is equivalent to $\bar{X}_J$: the logical gate with $\bar{X}$ on all the target logical qubits. This means$X^a Z^{a \cap \{\bar{U}^P\}} X^{a+b} = X^b Z^{a \cap \{\bar{U}^P\}}$ should be a stabilizer, which implies X^b is a stabilizer, contradicting that it is a logical X operator. □

3 Code Automorphisms and Addressability

In the previous section we applied quantum gates on physical qubits to implement logical quantum gates and showed limits of "simple" implementations. Another interesting gate would be the swap gate : exchanging two qubits together. In this section we observe the possible actions of swapping physical qubits of a code and prove some limits of such implementations.

Definition 3.1. *Let C a classical code with parity check G, $\tau_n \in S_n$ is an automorphism of C iff there exists $U \in Gl_r(\mathbb{F}_2)$ such that $UG = GP$ where P is the permutation matrix of τ_n.*

Definition 3.2. *$\tau_n \in S_n$ is an automorphism of $CSS(C_1, C_2)$ iff it is an automorphism of C_1 and C_2.*

In this case, calling $G = \begin{pmatrix} H_1 \\ H_2 \end{pmatrix}$ *where H_1, H_2 are the respective parity checks, we get that*

$UG = GP$ *with* $U = \begin{pmatrix} U_1 & 0 \\ 0 & U_2 \end{pmatrix}$.

Example 3.3. *Take* $H_1 = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}$ *and* $H_2 = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$, *we get* $G = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$.

We can see that $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$.

This means that the permutation of qubits represented by the matrix $\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$ *preserves the stabilizer group, meaning that it is a valid automorphism for the classical code represented by G but it is not a valid automorphism for $CSS(C_1, C_2)$ as it is not an automorphism on C_1, C_2.*

Remark 3.4. *The matrix* $G = \begin{pmatrix} H_1 \\ H_2 \end{pmatrix}$, *corresponds to the parity check matrix of the classical code where we forget if a check is of type X or Z.*

Intuitively, this means that an automorphism can permute the columns of G in such a way that it preserves the vector space. However this puts strong conditions on the form of the invertible matrix U.

Proposition 3.5. *The number of pairs U, P of invertible matrix and permutation such that $UG = GP$ only depends on the vector space of G and not the basis.*

Proof. Let $\bar{G} = WGP'$ be another basis of the vector space. Let U, P respectively an invertible matrix and a permutation, if $UGP^T = G$ then $(WUW^{-1})\bar{G}(P'^T PP') = \bar{G}$. Furthermore, WUW^{-1} is invertible and $P'^T PP'$ is a permutation. Thus, there are as many pairs for G and $\bar{G}$. □

Proposition 3.6. *Calling m the number of pairs U, P of invertible matrix and permutation such that $UG = GP$, and p the number of pairs of permutation such that $P'G' = G'P''$ we have $p \leq m \leq \frac{n!}{(n-r)!}$, where G' is the right block in the row reduced form of G.*

Proof. Since G has r independent rows, we can reduce it such that it has the form $\bar{G} = \begin{pmatrix} I_r & G' \end{pmatrix} = WGP'$ where $W \in GL_r(\mathbb{F}_2)$ and P' a permutation matrix of dimension n.

Using Proposition 3.5 we get that there are as many pairs for G and for $\bar{G}$.

Let M be an invertible matrix of rank r, we get that $M\bar{G} = \begin{pmatrix} M & MG' \end{pmatrix}$. Thus, for M to be part of a valid pair, it has to be made of r independent columns of $\bar{G}$. And there are less than $\frac{n!}{(n-r)!}$ ways to pick such set of columns.

For the lower bound, we can check that we can extend any valid permutation on G' into a valid permutation on $\bar{G}$. Let P', P'' be permutation matrix of dimension respectively $r, n-r$.

$$P'\bar{G} = \begin{pmatrix} P' & P'G' \end{pmatrix} = \begin{pmatrix} P' & G'P'' \end{pmatrix} = \begin{pmatrix} I_r & G' \end{pmatrix} \begin{pmatrix} P' & 0 \\ 0 & P'' \end{pmatrix} = \bar{G} \begin{pmatrix} P' & 0 \\ 0 & P'' \end{pmatrix}$$

Hence, any pair of permutation on G' can be extended into a valid one on $\bar{G}$.

□

Remark 3.7. *Furthermore, since we want $M\bar{G} = \bar{G}P$ with P a permutation of the columns, then we also need $M\bar{G}$ to generate I_r, meaning that the columns of M^{-1} should also be in $\bar{G}$. This is a stricter restriction and could reduce the upper bound significantly depending on the code. We use this observation to create an algorithm to find all matrices U having $UB = BP$ without going over all permutations. We present the algorithm and some of the results in Appendix D.2.*

Remark 3.8. *Here we are assuming that G is full rank, thus we can just consider G to be the parity check of a classical code. Or to apply this result directly to quantum codes, we need X and Z checks to be independent, and this hypothesis is not too restrictive.*

We are now going to use the upper bound on the number of automorphism of classical code, to describe families of CSS codes on which we cannot have addressable swaps implemented by only permutation of qubits.

Theorem 3.9. *Let $C_n = CSS(C_n^1, C_n^2)$ such that calling ρ_n', ρ_n'' the maximum and minimum of the rates of the classical codes C_n^1, C_n^2 we have $2\rho_n' + \rho_n'' > 2$ starting from some $n > n_1$. Then this family of codes does not have all addressable permutations of logical qubits implemented by permutations of physical qubits only.*

Proof. Let us consider only the codes for $n > n_1$. Let us assume that we have some permutation of the qubits P that preserves the codespace. This means that P is an automorphism for the CSS code, hence an automorphism for both classical codes that makes the CSS code. An upper bound of the number of such permutation P is thus the minimum of the number of automorphisms on X stabilizers and automorphisms on Z stabilizers. Calling ρ', ρ'' the maximum and minimum of the rates of the codes making the CSS code, we have using Proposition 3.6 that there is less than $\frac{n!}{(\rho' n)!}$ such automorphisms.

Furthermore, this CSS code encodes ρn where ρ is the rate of the code. Using Proposition 1.3 we have that $\rho = \rho' + \rho'' - 1$, thus $\rho' + \rho = 2\rho' + \rho'' > 2$ by assumption.

Now using Lemma C.1, we get that $\exists n_0 \in \mathbb{N}$ such that $\forall\, n \geq n_0$, $\frac{n!}{(\rho' n)!} < (\rho n)!$. Thus, starting from n_0, there is less possible automorphisms than permutation of the logical qubits. Meaning that we cannot have addressable swaps implemented by physical swaps for those families of asymptotically good rate codes.

□

Remark 3.10. *Since addressable CNOTs generate all the possible permutation of logical qubits. Then this theorem also says that we cannot implement all addressable CNOTs using swaps only on those codes.*

Corollary 3.11. *Let $(C_n)_{n\in\mathbb{N}}$ a family of CSS codes and $n_0 \in \mathbb{N}$ such that $\forall\, n > n_0,\ \rho_n > \frac{1}{3}$. Then this family of codes does not have addressable permutations of logical qubits implemented by permutations of physical qubits only.*

Proof. Calling C_n^1, C_n^2 the classical codes making C_n, and ρ_n', ρ_n'' the maximum and minimum of their rates, we get that $\rho_n = \rho_n' + \rho_n'' - 1$. Thus $\rho_n \leq 2\rho_n' - 1$ which means that $\rho_n' \geq \frac{\rho_n+1}{2}$. Hence $\rho_n + \rho_n' = 2\rho_n' + \rho_n'' \geq \frac{\rho_n+1}{2} + \rho_n > 1$. Thus, for all $n > n_0$, $2\rho_n' + \rho_n'' > 2$, and we can now use Theorem 3.9. □

Remark 3.12. *There is no limit to how small the rate of the CSS code is, we can construct codes verifying this hypothesis with ρ arbitrary close to* 0.

This result is interesting as it illustrates an example of a trade-off between the performances of codes and how easy it might be to implement some logical operations on them. In [PK22][LZ22] they prove methods to construct families of good quantum codes for any rate $0 < \rho < 1$, however using Theorem 3.9 we know that starting from $\rho > \frac{1}{3}$ implementing logical swaps is not possible with just physical swaps.

Corollary 3.13. *Let $C_n = CSS(C_n^1, C_n^2)$ such that calling ρ_n', ρ_n'' the maximum and minimum of the rates of the classical codes C_n^1, C_n^2 we have $2\rho_n' + \rho_n'' > 2$ starting from some $n > n_1$. Then this family of codes does not have all addressable logical CNOTs between two blocks implemented by transversal physical CNOTs on all qubits.*

Proof. In the case of CNOTs between two blocks the equations of validity are

$$\forall\, a \in A,\ \pi_I(a \cap I) \in A$$

$$\forall\, b \in B,\ \pi_J(b \cap J) \in B$$

Now let us take $I, J = [\![n]\!]$, meaning that all physical qubits participate in this implementation. We will denote such an implementation as $CNOT_\pi$ where π is the permutation associating a control qubit to its target. Using the equations, if $CNOT_\pi$ is a valid logical then $\forall\, a \in A,\ \pi(a) \in A$ and $\forall\, b \in B,\ \pi^{-1}(b) \in B$. This means that in this case, π is an automorphism of A and π^{-1} is an automorphism of B. Now as $\pi(B) = \pi(\pi^{-1}(B) = B$ we get that π is an automorphism for the CSS code.

Now if we want all addressable logical CNOTs, it means that for any logical CNOT$_{I\to J}$ that we can represent by its permutation π_I, we can implement it using the physical CNOTs described earlier. However there are $k!$ such logical CNOTs and less such physical CNOTs than automorphisms of the code.

Now by assumption we can use Lemma C.1, which gives us that from some $n_0 \in \mathbb{N}$, the number of automorphisms is smaller than $k!$. Hence, we cannot implement all addressable CNOTs between two blocks with transversal physical CNOTs on all qubits. □

4 Conclusion

While quantum error correcting codes are getting more and more efficient [PK22][LZ22], it is still unknown how we can apply gates efficiently on them. Work has been done to study transversal implementation of gates, but in the case of codes encoded on more than one qubit, they do not consider the question of applying logical gates to a subset of logical qubits. As quantum algorithms need those gates, it is important to study how we can do them efficiently.

In this work we studied the addressability problem on CSS code and proved that naive implementation of logical gates often requires a splitting code. We showed that on non-splitting codes any transversal implementation of addressable logical H, HP and PH gates must use non-Clifford gates. Using upper bounds on the number of automorphisms, we proved that any CSS code with asymptotical rate bigger than $\frac{1}{3}$ cannot have all addressable logical swaps implemented by physical swaps only, and cannot have all addressable logical CNOTs implemented by transversal physical

CNOTs involving all physical qubits. This is up to our knowledge the first result illustrating a trade-off between its performance and how easy it is to implement logical gates on it.

As mentioned before, up to our knowledge, this work is the first studying the addressability problem. As quantum computation needs efficient and fault tolerant implementation, we mainly restricted ourselves to Clifford gates and transversal implementations. In future work, it would be interesting to consider non-Clifford transversal implementations, or Clifford implementation while allowing bigger blocks, of bounded size, in the partition for transversality. Furthermore, as we could not identify a split caused by phase gates, it would be interesting to find codes with good parameters having addressable gates implemented by phases gates. Finally, in this study we only consider CSS codes which is a very general family of codes. Future work might want to look at more specific families of codes to identify more specific addressability results.

Acknowledgements : I would like to thank Samuel Jaques for his guidance and help during those 5 months, as well as the QKD team at IQC. It was an extremely enjoyable experience during which I learned a lot about quantum computing and error correcting codes.

References

[BB22] N. P. Breuckmann and Simon Burton. "Fold-Transversal Clifford Gates for Quantum Codes". In: *Quantum* (2022). URL: https://api.semanticscholar.org/CorpusID:246823565.

[Bur23] Burniston, John. "Pre-Privacy Amplification: A Post-Processing Technique for Quantum Key Distribution with Application to the Simplified Trusted Relay". MA thesis. 2023. URL: http://hdl.handle.net/10012/19329.

[Chu36] Alonzo Church. "A note on the Entscheidungsproblem". In: *Journal of Symbolic Logic* 1 (1936), pp. 40–41. URL: https://api.semanticscholar.org/CorpusID:42323521.

[CS96] A. R. Calderbank and Peter W. Shor. "Good quantum error-correcting codes exist". In: *Phys. Rev. A* 54 (2 Aug. 1996), pp. 1098–1105. DOI: 10.1103/PhysRevA.54.1098. URL: https://link.aps.org/doi/10.1103/PhysRevA.54.1098.

[EK09] Bryan Eastin and Emanuel Knill. "Restrictions on Transversal Encoded Quantum Gate Sets". In: *Phys. Rev. Lett.* 102 (11 Mar. 2009), p. 110502. DOI: 10.1103/PhysRevLett.102.110502. URL: https://link.aps.org/doi/10.1103/PhysRevLett.102.110502.

[Fey82] Richard P. Feynman. "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21.6 (June 1982), pp. 467–488. ISSN: 1572-9575. DOI: 10.1007/BF02650179. URL: https://doi.org/10.1007/BF02650179.

[Got98] D Gottesman. "The Heisenberg representation of quantum computers". In: (June 1998). URL: https://www.osti.gov/biblio/319738.

[Gro96] Lov K. Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866. URL: https://doi.org/10.1145/237814.237866.

[Hao21] Hanson Hao. "Investigations on Automorphism Groups of Quantum Stabilizer Codes". In: *CoRR* abs/2109.12735 (2021). arXiv: 2109.12735. URL: https://arxiv.org/abs/2109.12735.

[Kit03] A.Yu. Kitaev. "Fault-tolerant quantum computation by anyons". In: *Annals of Physics* 303.1 (2003), pp. 2–30. ISSN: 0003-4916. DOI: https://doi.org/10.1016/S0003-4916(02)00018-0. URL: https://www.sciencedirect.com/science/article/pii/S0003491602000180.

[LZ22] A. Leverrier and G. Zemor. "Quantum Tanner codes". In: *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2022, pp. 872–883. DOI: 10.1109/FOCS54457.2022.00117. URL: https://doi.ieeecomputersociety.org/10.1109/FOCS54457.2022.00117.

[PK22] Pavel Panteleev and Gleb Kalachev. "Asymptotically good Quantum and locally testable classical LDPC codes". In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2022. Rome, Italy: Association for Computing Machinery, 2022, pp. 375–388. ISBN: 9781450392648. DOI: 10.1145/3519935.3520017. URL: https://doi.org/10.1145/3519935.3520017.

[Sho95] Peter W. Shor. "Scheme for reducing decoherence in quantum computer memory". In: *Phys. Rev. A* 52 (4 Oct. 1995), R2493–R2496. DOI: 10.1103/PhysRevA.52.R2493. URL: https://link.aps.org/doi/10.1103/PhysRevA.52.R2493.

[Sho97] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172. eprint: https://doi.org/10.1137/S0097539795293172. URL: https://doi.org/10.1137/S0097539795293172.

[Ste96a] A. M. Steane. "Simple quantum error-correcting codes". In: *Phys. Rev. A* 54 (6 Dec. 1996), pp. 4741–4751. DOI: 10.1103/PhysRevA.54.4741. URL: https://link.aps.org/doi/10.1103/PhysRevA.54.4741.

[Ste96b] Andrew Steane. “Multiple-particle interference and quantum error correction”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 452.1954 (1996), pp. 2551–2577. DOI: 10.1098/rspa.1996.0136. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rspa.1996.0136. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1996.0136.

[Tur37] A. M. Turing. “On Computable Numbers, with an Application to the Entscheidungsproblem”. In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265. DOI: https://doi.org/10.1112/plms/s2-42.1.230. eprint: https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/plms/s2-42.1.230. URL: https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/plms/s2-42.1.230.

[WBB22] Mark A. Webster, Benjamin J. Brown, and Stephen D. Bartlett. “The XP Stabiliser Formalism: a Generalisation of the Pauli Stabiliser Formalism with Arbitrary Phases”. In: *Quantum* 6 (Sept. 2022), p. 815. ISSN: 2521-327X. DOI: 10.22331/q-2022-09-22-815. URL: https://doi.org/10.22331/q-2022-09-22-815.

[Zhu+23] Guanyu Zhu, Shehryar Sikander, Elia Portnoy, Andrew W. Cross, and Benjamin J. Brown. “Non-Clifford and parallelizable fault-tolerant logical gates on constant and almost-constant rate homological quantum LDPC codes via higher symmetries”. In: *ArXiv* abs/2310.16982 (2023). URL: https://api.semanticscholar.org/CorpusID:264490902.

A Additional background

A.1 Quantum computing

In 1982 Richard Feynmann [Fey82] conjectured that a quantum computer, that would use quantum mechanics at its base might be more efficient that a classical computer when it comes to simulating quantum systems. Since then problems were proved to be more efficiently computable using quantum mechanics such as the search among a list of N elements that can be done in $\mathcal{O}(\sqrt{N})$ using Grover algorithm [Gro96]. However, even if Shor's algorithm [Sho97] for the problem of the factorization is believed to be, we have yet to prove any exponential speedup using quantum. Such a result would refute the strong Church-Turing thesis [Chu36][Tur37] stating that all universal implementation of universal computation are equivalent up to polynomial time. Finally, proving that Shor's algorithm represents an exponential speedup would also imply $P \neq NP$.

While classical computing uses bits, that are either 0 or 1, quantum computing uses qubits, that are quantum states in a superposition between 0 and 1 : $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ such that $\alpha^2 + \beta^2 = 1$. When measuring the state $|\Psi\rangle$, it has a probability α^2 to give 0 and β^2 to give 1.

However the real benefit of using quantum mechanics comes from entanglement. For example let us say we have a two-qubits state $|\Psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, then when measuring the first qubit, we have equal probability for 0 and 1, but the second qubit will for sure give the same output : the two qubits are entangled. When a state $|\Psi\rangle$ is not entangled it is said to be separable and we can describe the global quantum state as the tensor product $|\Psi_A\rangle \otimes |\Psi_B\rangle$.

As mentioned before, a quantum state has to be normalized in order to be a valid state. Thus the operations we apply to them must preserve their norm. Hence, the quantum gates are modelled by unitaries : the n-qubits quantum gates are in $\mathcal{O}_{2^n}(\mathbb{C})$. The inverse of a unitary U is its transpose-conjugate noted $U^\dagger$. The following quantum gates are the most classic ones and are the base for most quantum computation.

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

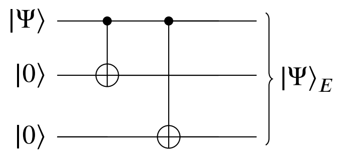
$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad P = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

The three first one together with the identity are the Pauli matrices and we can generate any unitary as a linear combination of tensor products of them. X corresponds to a bit flip, while Z is a phase flip, and $Y = iXZ$ can be seen as the two of them. Thus, we can use those gates to represent the quantum noise. There are two noise model that are commonly used : the Pauli error model and the depolarizing noise model. The first consists of assuming that an X is applied after each operator with probability p, same for Z, thus Y appears with probability p^2. The depolarizing noise model assumes that after each gate we apply a X or a Z or a Y, all with probability p.

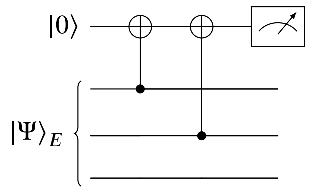
A.2 Quantum error correcting codes

A.2.1 Examples of QECC

Let us take a 3-repetition code as an example. In this code the logical 0 corresponds to the state $|000\rangle$ and the logical 1 to $|111\rangle$. Hence, the state $|\Psi\rangle = a|0\rangle + b|1\rangle$ will be encoded as $|\Psi\rangle_E = a|000\rangle + b|111\rangle$. Such an encoding can be done with the following circuit :



Now if we want to be able to decode, we can use a parity check circuit that give 0 if two qubits are in $|00\rangle$ or $|11\rangle$ and -1 when it is in $|10\rangle$ or $|01\rangle$. The following circuit checks the parity of the two first qubits.



Using this circuits we can do a majority vote, and assuming a single bit flip, we can correct the state. In this way, the encoding is a good way to protect an information that we are sending assuming that not too many errors happen during the transmission.

$$|\Psi\rangle \xrightarrow{Encoding} |\Psi\rangle_E \xrightarrow{Noise} |\tilde{\Psi}\rangle_E \xrightarrow{Decoding} |\Psi\rangle$$

Another interesting property is that we can implement a logical X gate by applying X to all physical qubits since $\mathrm{XXX}|000\rangle = |111\rangle$. Meaning that the X gate has a transversal implementation on this code.

A.2.2 Basics of error correcting codes

Definition A.1. *An error correcting code is an injective map : $Enc \; : \; \mathbb{F}^k \rightarrow \mathbb{F}^n$.*

- *$\mathbb{F}^k$ is the message space, and n is the block length.*
- *$\frac{k}{n}$ is the rate of the code and represents its efficiency.*
- *The code C is the image of the map and its elements are codewords.*
- *C is linear if the map is linear,*
- *d the distance of the code is the minimum Hamming weight of a non-identity logical operator. In the case of linear codes, it is the minimum Hamming weight of non trivial codewords.*

We denote a classical code by $[n, k, d]$, and a quantum code by $[\![n, k, d]\!]$.

For example, Shor's repetition code is a $[\![9, 1, 3]\!]$ code, whereas Kitaev toric code has parameter $[\![n, 1, \sqrt{n}]\!]$.

Theorem A.2. *Let C be a code with distance d, and e and error of Hamming weight s.*
Then we can detect e if $s < d$, and correct e if $s < t = \lfloor \frac{d-1}{2} \rfloor$.

Proof. As the distance between any two codewords is at least $d < s$ we get that the error is detectable as it does not send codewords to codewords. Now if the error has norm $s \leq t$ then we can correct it by going back to the closest codeword. □

Proposition A.3. *Let C be a linear code, then there exists a matrix H such that $C = Ker(H)$, we say that H is the parity check matrix of the code. This matrix H is also a basis for $C^\perp$ the dual code of C defined as the words being orthogonal to every codewords.*

Proof. Take H a basis of $C^\perp$ the orthogonal complement of C, this is an $(n - k) \times n$ matrix such that $C = \mathrm{Ker}(H)$. □

Remark A.4. *In the case of the field being $\mathbb{F}_2$, a code can be included in its dual or even be its own dual.*

Let $x \in \mathbb{F}_2^n$, we can write it as $x = c + e$ where c is a codeword and e and error. Using the parity check matrix H we have that $Hx = Hc + He = He$. We call the value He the syndrome of x, the goal of error correction, is given the syndrome, how can we deduct/correct the error.

B Adressability of P and CNOT gate on a single block

B.1 Phase gate

Proposition B.1. *Let C be a CSS code such that $S = \{X^a, Z^b \mid a \in A, b \in B\}$, with $A, B \subseteq \mathbb{Z}_2^n$. If P^h is a valid logical and $A = B$, then S splits on some support h and $|a \cap h| = 0 \mod 4$ for $a \in A$ iff P^h is a valid logical.*

Proof. Using the same analyses as for the H gate, let $a \in A$, then $PX^aP\dagger = X^{a\setminus h}Y^{a\cap h} = i^{|a\cap h|}X^aZ^{a\cap h}$. Thus for P^h to be a valid logical we need $a \cap h \in B$ and $|a \cap h| = 0 \bmod 4$. This means that if $A = B$ then P^h being a valid logical implies that $A = B$ splits on h and thus that S splits on h.

Now if $A = B$, considering that S splits on some support that we call h, we get that S^h is a valid logical iff for all $a_h \in A_h$ which is the restriction of A to the support of h, then $|a_h| = 0 \bmod 4$. □

The interesting result is that in the general case, we do not need any splitting for P^h to be a valid logical, hence we can have some addressable P gates without having a splitting code !

Remark B.2. *The fact that B splits does not mean that A splits. To see that let us take the code S such that :*

$$A = Span\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \end{pmatrix} \text{ and } B = Span\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

We can see that this code is valid since the X and Z stabilizers commute (ie $A \subseteq B^\perp$). However, B splits without A splitting ! Let us take this code but make each vector repeat 2 times (meaning that we double the number of qubits). Then for each vector in $b \in B$, we can see that P^b is a valid logical on this code. To see this, we can just observe that for all $a, b \in A, B$, $a \cap b \in B$ and $|a \cap b| = 0 \mod 4$.

A more general way to construct codes that have adressability on P would be to take two classical codes having parity checks H_A, H_B such that all parity checks have weight 4. We make $B = \begin{pmatrix} H_A & 0 \\ 0 & H_B \end{pmatrix}$ and $A = \begin{pmatrix} H_A & H_B \end{pmatrix}$. This is a valid CSS code that has an addressable P since applying P on each block is non trivial, and the code does not split. However the distance of such codes might not be optimal (even bad in most cases).

However , the distance of such codes might not be optimal. For example, let us say that H_A, H_B have dimension $k \times n_A, k \times n_B$. Take any x_A, x_B vectors of length $n_{A}, {}_B$. Let us call $\bar{d} = \min_{x_A, x_B \in (\mathbb{F}_2^{n_A}\setminus\{0\})\times(\mathbb{F}_2^{n_B}\setminus\{0\})}\{|x_A| + |x_B| \text{ , } H_Ax_A = H_Bx_B\}$. To explain this definition, let us consider x_A, x_B such that $H_Ax_A = H_Bx_B$. We have that $A(x_A\ x_B) = H_Ax_A + H_Bx_B$. Hence, if $H_Ax_A = H_Bx_B$ then $(x_A\ x_B)$ represent a valid logical operator. Now for this operator not to be in B we need it not to split, meaning that we just need to ask $x_A \neq 0, x_B \neq 0$. Hence $\bar{d}$ represent the smallest weight of such operators and this means that $d \leq \bar{d}$ with d the distance of the code. For example, if H_A, H_B share a columns then $\bar{d} \leq 2$. This gives an insight as to why having X of Z checks splitting and not the other might lead to codes with bad distances.

Example B.3. *Let us take*

$$H_A = \begin{pmatrix} 1&0&1&1&0&0&0&1\\ 1&0&1&0&1&1&0&0\\ 0&1&1&0&1&1&0&0\\ 0&1&1&1&1&0&0&0\\ 0&0&0&0&1&1&1&1 \end{pmatrix} \text{ and } H_B = \begin{pmatrix} 1&0&1&0&1&0&1&0\\ 0&1&0&0&1&0&1&1\\ 1&0&1&1&0&0&1&0\\ 0&0&1&0&1&1&0&1\\ 0&0&0&0&1&1&1&1 \end{pmatrix}$$

We can see that the classical codes C_A, C_B with parity check H_A, H_B have distance 3. Indeed, no two columns are similar and no column is made only of 0, so the distance is bigger than 2. Now consider $x_A = \begin{pmatrix} 1&1&1&0&0&0&0&0 \end{pmatrix}$ and $x_B = \begin{pmatrix} 0&1&0&0&0&1&0&1 \end{pmatrix}$ We have that $H_A x_A^T = 0$ and $H_B x_B^T = 0$, thus they represent non trivial codewords and have weight 3. Hence $2 < d \leq 3$ which gives $d = 3$ for C_A and C_B.

Using the same analysis, we find that the code with parity check $A = \begin{pmatrix} H_A & H_B \end{pmatrix}$ has distance 3. And when we consider $B = \begin{pmatrix} H_A & 0 \\ 0 & H_B \end{pmatrix}$ the distance of the associated code is also 3. Hence, the distance of the CSS code CSS(A,B) is 3 and it has addressable P gate.

However, we can check that in this example both logical operators implemented with P on each block are logical identities.

B.2 CNOT gate on the same block

After studying the H and P gates, it is natural to look at the $CNOT$ gate as those 3 gates generates the group of Clifford gates.

Let us first consider the case where we apply the CNOT on one block : both qubits are from the same block of code. We define $CNOT^{(i,j)_R}$ as the gate that does $CNOT(i,j)$ for all pair $(i,j) \in R \subseteq I \times J$. We call I the controls and J the targets. Since we want parallelized gates we add the condition that $I \cap J = \emptyset$ and that R is a bijective relation from I to J, this conditions will not be there in the case of CNOTs between two blocks.

We denote by π_R the function such that

$$\begin{cases} \forall\, i \in I, \pi_R(i) = j \text{ such that } (i,j) \in R \\ \forall\, j \in J, \pi_R(j) = i \text{ such that } (i,j) \in R \\ \forall x \notin I \cup J, \pi_R(x) = x \end{cases} \tag{4}$$

We denote by $\pi_R(u)$ the string $(v_k)_{k\leq n}$ where $v_k = u_{\pi_R^{-1}(k)}$. Furthermore, it is easy to observe that $\pi_R^2 = Id$.

Proposition B.4. *Let C be a CSS code such that $S = \{X^a, Z^b \mid a \in A, b \in B\}$, with $A, B \subseteq \mathbb{Z}_2^n$. If $CNOT^{(i,j)_R}$ is a valid logical then*

$$\begin{cases} \forall\, a \in A, \pi_R(a \cap I) \in A \\ \forall\, b \in B, \pi_R(b \cap J) \in B \end{cases} \tag{5}$$

Proof. Let X^a be a stabilizer, then $CNOT^{(i,j)_R} X^a (CNOT^{(i,j)_R})^\dagger = X^{a \oplus \pi_R(a\cap I)}$. Thus as X^a is a valid stabilizer, and the stabilizer group is stable by product, we get that $X^{a\oplus\pi_R(a\cap I)}$ is a stabilizer iff $\forall\, a \in A, \pi_R(a \cap I) \in A$. Doing the same analysis on B while inverting control that is the support of the Z stabilizer gives the other equation. □

Corollary B.5. *Let C be a CSS code such that $S = \{X^a, Z^b \mid a \in A, b \in B\}$, with $A, B \subseteq \mathbb{Z}_2^n$. If $A = B$, then $CNOT^{(i,j)_R}$ being a valid logical implies that S splits on I and J.*

Proof. Using the equations from Proposition B.4, we get that if $A = B$ and those CNOTs form a valid logical, then taking $a \in A$ we have $\pi_R(a \cap I) \in A = B$. Applying the equation for B we get $\pi_R((\pi_R(a \cap I) \cap J) \in B$. However $\pi_R(a \cap I)$ has support in J thus $\pi_R((\pi_R(a \cap I) \cap J) = a \cap I$. Hence, $\forall\, a \in A, a \cap I \in A$ and for the same reasons we can start from B to get the split on J. □

Remark B.6. *In the case of general CSS code, having a valid logical CNOT does not imply any splitting, for example take* $A = Span\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$

With $R = (i, i+4)_{1\leq i\leq 4}$, we get that $\pi_R(a \cap I) \in A$ for all $a \in A$. And we can just construct B by putting the 4 right qubits on the left, which satisfies the equation for B. Finally, those blocks do not split.

B.3 Proof for transversal Clifford implementation of PH and HP

Theorem B.7. *Any implementation of addressable logical HP using transversal Clifford requires a splitting code. Furthermore, the logical HP is applied to all logical qubits of a subcode and implemented using physical HP on all qubits of this subcode.*

Any implementation of addressable logical PH using transversal Clifford requires a splitting code. Furthermore, the logical PH is applied to all logical qubits of a subcode and implemented using physical PH on all qubits of this subcode.

Proof. Let us start we the case of HP which correspond to the type 2 single qubit Clifford. This means that considering again the minimal representatives, and U a transversal implementation of HP:

$\bar{U}X^{\tilde{a}_i}\bar{U}^\dagger \equiv X^{\tilde{a}_i}Z^{\tilde{b}_i}$. This gives using Remark 2.17 that $\tilde{a}_i \cap \{\bar{U}^{PH,H}\} = 0$, which means $\tilde{a}_i \subseteq \{\bar{U}^{I,P,HP,PHP}\}$. Furthermore, since U implements a logical HP we get that $\bar{U}Z^{\tilde{b}_i}\bar{U}^\dagger \equiv X^{\tilde{a}_i}$ which implies by Proposition 2.16 that $\tilde{b}_i \subseteq \{\bar{U}^{HP,H}\}$.

Now using that U is a valid logical, we get by Corollary 2.14 that the B block splits on $\{\bar{U}^{HP}\}$, $\{\bar{U}^{PH}\}$ and $\{\bar{U}^{H}\}$. This implies that the X logicals also split on those blocks. Meaning that any minimal representative of an X logical is either in $\{\bar{U}^{HP}\}$ or $\{\bar{U}^{PH}\}$ or $\{\bar{U}^{H}\}$ or $\{\bar{U}^{I,P,PHP}\}$. However since $\tilde{a}_i$ and $\tilde{b}_i$ must have a non empty intersection of their support, and that to implement a logical HP on logical qubit i we need $\tilde{b}_i \subseteq \{\bar{U}^{HP,H}\}$, it gives that $\tilde{a}_i \subseteq \{\bar{U}^{HP}\}$ and $\tilde{b}_i \subseteq \{\bar{U}^{HP}\}$ as A splits on $\{\bar{U}^{HP}\}$. Hence, a logical HP implemented by transversal Clifford gates must be implemented using physical HP on the subcode defined by the qubits in $\{\bar{U}^{HP}\}$.

Using the exact same reasoning we can prove a similar result for PH which would gives that logical PH implemented by transversal Clifford gates must be implemented using physical PH on the subcode defined by the qubits in $\{\bar{U}^{PH}\}$.

□

C Automorphisms

Lemma C.1. *For all $\rho + \rho' > 1$,*

$$\exists n_0 \in \mathbb{N} \text{ such that } \forall\, n \geq n_0,\ \frac{n!}{(\rho' n)!} < (\rho n)!$$

Proof. Let us consider the logarithm of this inequality, it gives : $\sum_{\rho' n\leq i\leq n}\log(i) < \sum_{2\leq j\leq \rho n}\log(j)$. Using that we are in the positive part of logarithm and that it's increasing then we can say :

$$\sum_{\rho' n\leq i\leq n}\log(i) \leq \int_{\rho' n}^{n+1}\log(x)dx$$

$$\sum_{2\leq i\leq \rho n}\log(i) \geq \int_{1}^{\rho n-1}\log(x)dx$$

Now using $\int_{\rho' n}^{n+1} log(x)dx = g(n+1)-g(\rho' n)$ where $g(x) = x\log(x)-x$. And $\int_{1}^{\rho n-1}\log(x)dx = g(\rho n-1)-g(1)$. Hence, if $g(n+1)-g(\rho' n) < g(\rho n-1)-1$ then the inequality holds.

$$g(n+1)-g(\rho' n)-g(\rho n-1)+g(1) = (1-\rho'-\rho)n\log(n)+\mathcal{O}(n)$$

Since we assumed $\rho'+\rho > 1$, it gives that there exists an n_0 for which $\forall\, n \geq n_0$, $\frac{n!}{(\rho' n)!} < (\rho n)!$ □

C.1 Addressability with fold transversality

Definition C.2. *[BB22] $\tau_n \in S_n$ is a ZX-duality for $CSS(C_1, C_2)$ iff it is an automorphism of G and U such that $UG = GP$ is such that*

$$UG = GP \text{ with } U = \begin{pmatrix} 0 & U_2 \\ U_1 & 0 \end{pmatrix}.$$

Intuitively, when calling $C = CSS(A, B)$ a ZX-duality is a permutation sending A to B and B to A.

Remark C.3. *The concept of ZX duality is actually a restriction of what [Hao21] called Clifford automorphisms that are defined for C a stabilizer code with S the stabilizer group as*

$$Aut_{Cliff}(C) = \{\tau_n \in S_n \mid \tau_n(S) = \lambda_\sigma S \lambda_\sigma^{-1}\}$$

where λ_σ is a tensor product of n single qubit Clifford gates. It is quite easy to see that ZX duality are the permutation in $Aut_{Cliff}(C)$ such that $\lambda_\sigma = H^{\otimes n}$.

Proposition C.4. *Let $C = CSS(A, B)$, let us assume that C is a self-ZX-dual code, such that $U = H^h \bigotimes_{i\langle\tau(i)} SWAP_{i,\tau(i)}$ is a valid logical, then S splits on h.*

Proof. As the code is self-ZX-dual it means that there exists a permutation on the qubits τ_n such that $\tau_n(A) = B$ and $\tau_n(B) = A$. And as $\tau(n)$ is a bijection, we get that $dim(B) = dim(\tau_n(A)) = dim(A)$. The unitary U being a valid logical means that $\forall\, a \in A, \tau_n(a) \cap h \in B$ and $\tau_n(a)\backslash h \in A$. However, as $\tau_n(A) = B$, we get that every element $b \in B$ can be seen as $\tau_n(a)$ for some $a \in A$, thus $\forall\, b \in B, b \cap h \in B$. Now doing the same by starting on B gives the same splitting on A, thus U being a valid logical means that S splits on h.

□

D Algorithms

D.1 Deciding if a code splits

In his master thesis, John Burniston [Bur23] develops a method to identify if a matrix is "reducible" where F being reducible means that it either has a column of zeros or that there is an invertible map U and a permutation P such that $F = \begin{pmatrix} F_1 & 0 \\ 0 & F_2 \end{pmatrix}$. Hence, in our case, the definition of reducibility is the same as the definition of split of the X or Z checks. If X, Z checks are reducible, and if they share a common split, then taking the common blocks gives us that the stabilizer group splits and that the code represents a splitting code.

The following algorithms follows his method to find if a matrix is reducible : that is the Find-Blocks algorithm. Finally, we use another graph representation and connected components to find the common blocks division between X and Z stabilizers, thus giving the split of the code.

Given generators of its stabilizer group represented as a $r \times n$ matrix, this algorithm returns in time $\mathcal{O}(r \times n)$ the split decomposition of a CSS code.

Algorithm 1 Common Block Diagonalization

```
Require: Stabilizer matrix $S = \begin{bmatrix} S_X & 0 \\ 0 & S_Z \end{bmatrix}$
Ensure: Partition of columns for common block diagonalization
1: $B_X \leftarrow$ Find_blocks($S_X$)
2: $B_Z \leftarrow$ Find_blocks($S_Z$)
3: $P \leftarrow$ Common_blocks($B_X, B_Z$)
4: return $P$
```

Algorithm 2 Find_blocks

```
Require: Matrix $F$
Ensure: Set of columns for each block in block diagonal form
1: $F' \leftarrow$ RowReducedForm($F$) { Ensures $F' = \begin{bmatrix} I_r & F'' \end{bmatrix}$ }
2: $G \leftarrow$ TannerGraph($F'$)
3: $C \leftarrow$ ConnectedComponents($G$)
4: Blocks $\leftarrow$ BlocksFromConnectedComponents($C$)
5: return Blocks
```

Algorithm 3 TannerGraph

```
Require: Matrix $F$
Ensure: Bipartite graph $G$ with edges between $r_i$ and $c_j$ if $F_{ij} = 1$
1: Initialize graph $G = (R, C, E)$ where $R$ are row vertices, $C$ are column vertices, and $E$ are
   edges
2: for each entry $F_{ij}$ in $F$ do
3:     if $F_{ij} = 1$ then
4:         Add edge $(r_i, c_j)$ to $E$
5:     end if
6: end for
7: return $G$
```

D.2 Unitaries implementable by permutations

This algorithm computes in $\mathcal{O}\left(\frac{n!}{r!(n-r)!}\right)$, where B is an $r \times n$ matrix, all the logical gates implementable with physical swaps on the classical code with B the parity check matrix.

Algorithm 4 BlocksFromConnectedComponents

```
Require: Connected components $C$
Ensure: Column index for the block diagonalization of the matrix
1: Blocks $\leftarrow \emptyset$
2: for $C_i$ in $C$ do
3:     block$_i \leftarrow \emptyset$
4:     for $c_j$ column vertex in $C_i$ do
5:         block$_i \leftarrow$ block$_i \cup \{j\}$
6:     end for
7:     Blocks $\leftarrow$ Blocks $\cup$ {block$_i$}
8: end for
9: return Blocks
```

Algorithm 5 Common_blocks

```
Require: Block partitions $B_X$ and $B_Z$
Ensure: Common block partition $P$
1: $G \leftarrow (V, E)$ where (Block$_i^X$, Block$_j^Z$) $\in E$ if Block$_i^X \cap$ Block$_j^Z \neq \emptyset$
2: $C \leftarrow$ ConnectedComponents($G$)
3: Blocks2sum $\leftarrow$ BlocksFromConnectedComponents($C$)
4: $P \leftarrow \emptyset$
5: for Blocks in Blocks2sum do
6:     Union $\leftarrow \emptyset$
7:     for Block in Blocks do
8:         Union $\leftarrow$ Union $\cup$ Block
9:     end for
10:    $P \leftarrow P \cup$ {Union}
11: end for
12: return $P$
```

Algorithm 6 Logical operations implementable by physical swaps

```
Require: Basis $B$
Ensure: All matrices $U$ such that $UB = BP$ for $P$ permutation
1: $W, B' \leftarrow$ Gaussian_Elimination($B$) {Ensure $B'$ is in the form $(I_r\ |*)$}
2: $S \leftarrow$ Find_Independent_Column_Sets($B'$) {Find all sets of $r$ independent columns in $B'$ up to
   permutations}
3: Operations $\leftarrow \emptyset$
4: for all $U \in S$ do
5:     if $U^{-1}$ has columns from $B'$ then
6:         if $UB' = B'P$ for some permutation $P$ then
7:             Operations $\leftarrow$ Operations $\cup \{W^{-1}UW\}$
8:         end if
9:     end if
10: end for
11: return Operations
```

Example D.1. *Let us take the following parity check matrix :*

$$B = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

The algorithm returns the following logical operators as the ones being implementable by permutations.

$$U_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad U_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad U_3 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad U_4 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$