Stabilizer Codes Implemented using Maple

Dissertation by
Dimitris Kakofengitis

Supervised by
Dr Terry G Rudolph

In partial Fulfillment of the Requirements
for the Degree of
Master of Science in
Quantum Field Theory and Fundamental Forces

Theoretical Physics Group

Imperial College London
South Kensington, London

Submitted : October 2, 2009

This page intentionally left blank.

To my friend Cecilia N Gyansah
who has supported me throughout this year.

This page intentionally left blank.

Contents

List of Tables

List of Figures

List of Algorithms

Preface

Stabilizer codes provide a better description of pure and mixed quantum states of N-qubit systems. Their applications have been mainly focused on quantum error correcting codes, where stabilizer states take the role of projectors on subspaces.

In this dissertation we employ elementary operations to present a number of normal forms as discussed in [1], for pure stabilizer states, together with descriptions of algorithms that allow the reduction of these normal forms. Some examples are presented in the programming language of Maple(at the time of this project, the latest version was Maple 13 which was released in April 2009).

Dimitris Kakofengitis
October 2, 2009

Chapter 1

Introduction

1.1 Introduction to Quantum Information

In quantum information, a *qubit* or quantum bit is the quantum analogue of the classical bit. A qubit is described by a state vector also known as *single-qubit state* which is a linear superposition of the basis states, $|0\rangle$ and $|1\rangle$,

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1.1}$$

that is a *normalized* state, where α and β are complex numbers, with $|\alpha|^2 + |\beta|^2 = 1$. A representation of a single-qubit state can be shown in a Bloch sphere, see Figure 1.1. The *canonical* parametrization of the single-qubit

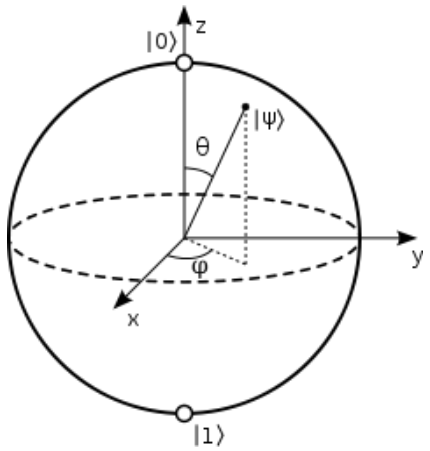

Figure 1.1: Bloch sphere, a geometrical representation of two-level quantum system.

state can also be written as,

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle \tag{1.2}$$

where θ and ϕ can take values between, $0 \leq \theta \leq \pi$ and $0 \leq \varphi \leq 2\pi$.

The Bell state, $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, is an example of an *entangled* state. Entangled states cannot be written as the product of two single-qubit states and are graphically positioned inside the Bloch sphere, whereas single-qubit states are positioned at the shell of the Bloch sphere.

Making a measurement on state 1.1 by applying a projection operator, the state collapses with probability $|\alpha|^2$ to the outcome $|0\rangle$ and with probability $|\beta|^2$ to the outcome $|1\rangle$. Since state 1.1 is normalized then the sum of the two probabilities will be exactly 1.

The Pauli operators,

$$\begin{aligned}
\sigma_1 = \sigma_x = X &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \\
\sigma_2 = \sigma_y = Y &= \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \\
\sigma_3 = \sigma_z = Z &= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},
\end{aligned} \tag{1.3}$$

where i is the imaginary number equal to the square root of -1, are an example of projection operators on single-qubit states and are considered as spin matrices as they represent an observable describing the spin of a spin $\frac{1}{2}$ particle in the three spatial directions. Note that $\sigma_y = i\sigma_x\sigma_z$. Such operators are said to be *Hermitian* since the *adjoint*, which is the complex conjugate transpose of the operator is equal to the operator itself such that,

$$\sigma_i^\dagger = \sigma_i$$

where $i \in \{x, y, z\}$. This also implies that,

$$\sigma_i^\dagger\sigma_i = \sigma_i\sigma_i^\dagger = \mathbb{1},$$

$$\text{where } \mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

and is called the identity matrix. Another important algebraic property of the Pauli operators is the *commutation* and *anti-commutation* relations which will be used later as a proof of legitimacy of the stabilizer arrays. The Pauli operators anti-commute, that is for $i \neq j$, for $i, j \in \{x, y, z\}$,

$$\{\sigma_i, \sigma_j\} = \sigma_i\sigma_j + \sigma_j\sigma_i = 0.$$

In this dissertation we will make use of the commutation relation,

$$[A, B] = AB - BA$$

of the Pauli operators shown in Table 1.1.

	σ_x	σ_y	σ_z
σ_x	0	$2i\sigma_z$	$-2i\sigma_y$
σ_y	$-2i\sigma_z$	0	$2i\sigma_x$
σ_z	$2i\sigma_y$	$-2i\sigma_x$	0

Table 1.1: Commutation relations for the Pauli operators.

1.2 Stabilizer Formalism

Stabilizer codes also known as quantum error correcting codes, have been introduced first by D. Gottesman in his Thesis, *Stabilizer Codes and Quantum Error Correction* (see [3] for more details) and are a more compact way of presenting states such as the *Calderbank-Shor-Steane* codes, known as *CSS* codes after the initials of their inventors. These stabilizer states or stabilizer sets are formed by tensor product operations of Hermitian operators, such as the Pauli operators. The fact that an n-particle stabilizer state is determined as the joint unique eigenvector with eigenvalue $+1$ of a set of only N tensor products of Pauli operators, allows for a more detailed study of their entanglement properties. In this context stabilizer formalism is used as a powerful tool for the efficient description of pure and mixed quantum states of N-qubit systems.

In order to understand the meaning of the word stabilizer, suppose an operator A stabilizes a subspace S, when for all states $|\psi\rangle \in S$,

$$A|\psi\rangle = |\psi\rangle . \tag{1.4}$$

In other words, $|\psi\rangle$ is an eigenstate of A with eigenvalue $+1$. Table 1.2 shows the states which are stabilized by the Pauli operators. Another way of illustrating stabilizer formalism is by using the Bell states,

$$|\Phi^+\rangle = \tfrac{|00\rangle+|11\rangle}{\sqrt{2}}, |\Phi^-\rangle = \tfrac{|00\rangle-|11\rangle}{\sqrt{2}},$$
$$|\Psi^+\rangle = \tfrac{|01\rangle+|10\rangle}{\sqrt{2}} \text{and} |\Psi^-\rangle = \tfrac{|01\rangle-|10\rangle}{\sqrt{2}}.$$

State $|\Phi^+\rangle$ satisfies the identities $X_1X_2|\Phi^+\rangle = |\Phi^+\rangle$ and $Z_1Z_2|\Phi^+\rangle = |\Phi^+\rangle$, state $|\Phi^-\rangle$ satisfies $-X_1X_2|\Phi^-\rangle = |\Phi^-\rangle$ and $Z_1Z_2|\Phi^-\rangle = |\Phi^-\rangle$, state $|\Psi^+\rangle$

$\|\psi\rangle$	A
$\|0\rangle + \|1\rangle$	X
$\|0\rangle - \|1\rangle$	$-X$
$\|0\rangle + i\,\|1\rangle$	Y
$\|0\rangle - i\,\|1\rangle$	$-Y$
$\|0\rangle$	Z
$\|1\rangle$	$-Z$

Table 1.2: Examples of state $|\psi\rangle$ which obey Equation 1.4 and their corresponding stabilizing operators.

satisfies $X_1X_2\,|\Psi^+\rangle = |\Psi^+\rangle$ and $-Z_1Z_2\,|\Psi^+\rangle = |\Psi^+\rangle$, state $|\Psi^-\rangle$ satisfies $-X_1X_2\,|\Psi^-\rangle = |\Psi^-\rangle$ and $-Z_1Z_2\,|\Psi^-\rangle = |\Psi^-\rangle$. Therefore the Bell states are stabilized by the operators X_1X_2, $-X_1X_2$, Z_1Z_2 and $-Z_1Z_2$, as shown in Table 1.3, up to a *global phase*. What is essentially shown here is that many quantum states can be described by working with the operators that stabilize them. Note that in this notation, X_1 is the operator X acting on the first

Bell State	Operators
$\|\Phi^+\rangle$	X_1X_2, Z_1Z_2
$\|\Phi^-\rangle$	$-X_1X_2$, Z_1Z_2
$\|\Psi^+\rangle$	X_1X_2, $-Z_1Z_2$
$\|\Psi^-\rangle$	$-X_1X_2$, $-Z_1Z_2$

Table 1.3: The Bell states and their corresponding operators, which stabilize them.

qubit and X_2 is the same operator acting on the second qubit. Also note that one operator stabilizes two different Bell states like the Bell states $|\Psi^+\rangle$ and $|\Phi^+\rangle$ have one common operator, X_1X_2.

The stabilizer formalism is primarily based on *Pauli group* G_n on N qubits. The Pauli group G_1 on a single qubit is the group consisting of $\mathbb{1}$ and all the Pauli operators (X,Y,Z), together with the multiplicative factors $\pm\mathbb{1}$, $\pm i$,

$$G_1 \equiv \{\pm\mathbb{1}, \pm i\mathbb{1}, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\}\,.$$

This set of matrices forms a group under the operation of matrix multiplication.

Suppose S is a subgroup of G_n and define V_s to be the set of N qubit states which are fixed by every element of S. V_s is the *vector space stabilized*

by S and S is said to be *stabilizer* of the space V_s since every element of V_s is stable under the action of elements in S.

For the case where $n = 3$ qubits and $S_1 \equiv \{\mathbb{1}, Z_1Z_2, Z_2Z_3, Z_1Z_3\}$, the subspace fixed by Z_1Z_2 is spanned by $|000\rangle$, $|001\rangle$, $|110\rangle$ and $|111\rangle$ and the subspace fixed by Z_2Z_3 is spanned by $|000\rangle$, $|100\rangle$, $|011\rangle$ and $|111\rangle$. The subspace spanned by the states $|000\rangle$ and $|111\rangle$ is V_{S_1}. Another example for $n = 3$ qubits is $S_2 \equiv \{\mathbb{1}, X_1X_2, X_2X_3, X_1X_3\}$. The subspace fixed by X_1X_2 is spanned by $|+++\rangle$, $|++-\rangle$, $|--+\rangle$ and $|---\rangle$ and the subspace fixed by X_2X_3 is spanned by $|+++\rangle$, $|-++\rangle$, $|+--\rangle$ and $|---\rangle$, where $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. The subspace spanned by the states $|+++\rangle$ and $|---\rangle$ is V_{S_2}.

A more *compact* way of describing a group is by using *generators*. As explained in Appendix A, a set of elements $g_1....g_K$ in a group G is said to generate the group G if every element of G can be written as a product of elements from the list $g_1...g_K$ and we write $G = \langle g_1...g_K\rangle$. For the last two cases of $n = 3$ qubits, $S_1 = \langle Z_1Z_2, Z_2Z_3\rangle$ and $S_2 = \langle X_1X_2, X_2X_3\rangle$ as $Z_1Z_3 = (Z_1Z_2)(Z_2Z_3)$, $X_1X_3 = (X_1X_2)(X_2X_3)$ and $\mathbb{1} = (Z_1Z_2)^2 = (X_1X_2)^2$. In order to see that a particular vector is stabilized by a group S we need only to check that the vector is stabilized by the generators, making this a most convenient representation [2].

Two conditions have to be met so that a subgroup S of the Pauli group can be used as the stabilizer for a non-trivial vector space [2]:

a) The elements of S commute

b) $-\mathbb{1}$ is not an element of S

For example, let M and N be elements of S. Then M and N are tensor products of Pauli matrices. By assumption $-NM = MN$ so we have $-|\psi\rangle = -NM|\psi\rangle = MN|\psi\rangle = |\psi\rangle$, following from the fact that M and N stabilize $|\psi\rangle$. Therefore $-|\psi\rangle = |\psi\rangle$, which implies that $|\psi\rangle$ is the zero vector and therefore this is a contradiction to the first condition. As a contradiction to the second condition, note that if $-\mathbb{1}$ is an element of S then $-\mathbb{1}|\psi\rangle = |\psi\rangle$.

An example of the stabilizer formalism is the seven qubit *Steane code* [2], which is given by the state vectors,

$$\begin{aligned} |0_L\rangle &= \tfrac{1}{\sqrt{8}}[|0000000\rangle + |1010101\rangle + |0110011\rangle + |1100110\rangle \\ &\quad + |0001111\rangle + |1011010\rangle + |0111100\rangle + |1101001\rangle] \\ |1_L\rangle &= \tfrac{1}{\sqrt{8}}[|1111111\rangle + |0101010\rangle + |1001100\rangle + |0011001\rangle \\ &\quad + |1110000\rangle + |0100101\rangle + |1000011\rangle + |0010110\rangle]. \end{aligned} \tag{1.5}$$

The six generators g_1 through g_6 shown in Table 1.4, generate a stabilizer for the code space of the Steane code, which is a more compact way of describing

the code with relation to the specification in terms of state vectors Equation 1.5.

Name	Operator
g_1	$\mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1} \otimes X \otimes X \otimes X \otimes X$
g_2	$\mathbb{1} \otimes X \otimes X \otimes \mathbb{1} \otimes \mathbb{1} \otimes X \otimes X$
g_3	$X \otimes \mathbb{1} \otimes X \otimes \mathbb{1} \otimes X \otimes \mathbb{1} \otimes X$
g_4	$\mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1} \otimes Z \otimes Z \otimes Z \otimes Z$
g_5	$\mathbb{1} \otimes Z \otimes Z \otimes \mathbb{1} \otimes \mathbb{1} \otimes Z \otimes Z$
g_6	$Z \otimes \mathbb{1} \otimes Z \otimes \mathbb{1} \otimes Z \otimes \mathbb{1} \otimes Z$

Table 1.4: Stabilizer generators for the Steane seven qubit code. Note that the symbol $\otimes$ can be omitted for simplicity.

1.3 Stabilizer Operator

A *stabilizer operator* on N qubits is a tensor product of the operators taken from the set of Pauli operators (1.3) and the identity matrix $\mathbb{1}$. For $K = N$ a generator set $G = \{g_1 ... g_K\}$ uniquely determines a single state $|\psi\rangle$ that satisfies $g_k |\psi\rangle = |\psi\rangle$ for all $k = 1, \ldots, N$. Any state for which such a generator set exists is called *stabilizer state*. Such a state has trivially the property that $g_k |\psi\rangle \langle\psi| = |\psi\rangle \langle\psi|$ for all k such that [1],

$$|\psi\rangle \langle\psi| = \prod_{k=1}^{N} \frac{\mathbb{1} + g_k}{2}. \tag{1.6}$$

Considering two parties A and B, the reduced density matrix of the stabilizer state can be computed as [1]

$$\rho_A = Tr_B \rho = \frac{1}{2^N} \sum_{g \in S} Tr_B g. \tag{1.7}$$

which means that only operators g contribute, that have identity operators acting on all qubits belonging to B.

1.4 Mixed Stabilizer States

In the case of mixed states characterized by mixed stabilizer states, one simply considers sets of G that are linearly dependent. As a consequence,

by multiplying stabilizers, some of them become identical to $\mathbb{1}$ and only K linearly dependent ones remain. Then the common eigenspace of these K operators will have a dimension larger than 1. The density operator is again just the projector onto this eigenspace, rescaled to trace 1 [1],

$$\wp = \prod_{k=1}^{N} \frac{\mathbb{1} + g_k}{2}$$

$$\rho = \frac{1}{2^{N-K}} \wp \tag{1.8}$$

Given that $\wp$ is a projector onto a subspace of dimension 2^{N-K}, the entropy of ρ is simply $N - K$ [2].

1.5 Stabilizer Arrays

A *stabilizer array* is a rectangular array of K rows and N columns, where the elements are Pauli operators (1.3) or the Identity matrix. Table 1.5 shows a general form of a rectangular array which corresponds to a generator set $G = \{g_1 \cdots g_K\}$ on N qubits, The element in the k-th row and n-th column

g_1^1	g_1^2	$\cdots$	g_1^N
g_2^1	g_2^2	$\cdots$	g_2^N
$\vdots$	$\vdots$	$\vdots$	$\vdots$
g_{K-1}^1	g_{K-1}^2	$\cdots$	g_{K-1}^N
g_K^1	g_K^2	$\cdots$	g_K^N

Table 1.5: A Stabilizer array corresponding to a generator set $G = \{g_1 \cdots g_K\}$ on N qubits.

is the n-th tensor factor (corresponding to qubit n) of the k-th generator g_k. In some applications it is necessary to deal with generator phase factors, $\pm 1, \pm i$, from which only ± 1 make sense for the case of stabilizer states since are Hermitian. Phase factors are stored in a K-dimensional vector s, where s_k is the phase factor of generator g_k.

1.6 Elementary Operations [1]

The allowed elementary operations for transforming a *stabilizer array* are divided in two kinds, the row and column operations. In the case of row

operations, the stabilizer state is not altered but only the generator set. The *row transposition* interchanges two rows in the stabilizer array and the *row multiplication* multiplies one row with another one. The multiplication table for Pauli operators is shown in Table 1.6.

	$\mathbb{1}$	X	Y	Z
$\mathbb{1}$	$\mathbb{1}$	X	Y	Z
X	X	$\mathbb{1}$	iZ	$-iY$
Y	Y	$-iZ$	$\mathbb{1}$	iX
Z	Z	iY	$-iX$	$\mathbb{1}$

Table 1.6: Multiplication table for Pauli operators.

The second kind are the column operations, which may alter the state. *Single-qubit* operations act on one given column by permuting the Pauli operators among themselves. These operations can be constructed from combinations of Hadamard gates (H) and $\Pi/4$ gates (P). Table 1.7 presents the truth table for single-qubit operations employed by the CNF algorithm.

X	Y	Z	Unitary
X	Y	Z	$\mathbb{1}$
Z	X	Y	PH
Y	Z	X	$HP^\dagger$
$-X$	Z	Y	$PHP^\dagger$
Y	X	$-Z$	$HPPHP^\dagger$
Z	$-Y$	X	H

Table 1.7: Truth table for the single-qubit operations employed by the CNF algorithm.

Transposing two columns in the bipartite case is only allowed when both columns (qubits) belong to the same party. The CNOT gate between two qubits, one being the control qubit and one the target qubit, operates on the two corresponding columns of the stabilizer array. Table 1.8 shows the CNOT gate employed by the CNF algorithm.

C	T	C'	T'
$\mathbb{1}$	$\mathbb{1}$	$\mathbb{1}$	$\mathbb{1}$
$\mathbb{1}$	X	$\mathbb{1}$	X
$\mathbb{1}$	Y	Z	Y
$\mathbb{1}$	Z	Z	Z
X	$\mathbb{1}$	X	X
X	X	X	$\mathbb{1}$
X	Y	Y	Z
X	Z	$-Y$	Y

C	T	C'	T'
Y	$\mathbb{1}$	Y	X
Y	X	Y	$\mathbb{1}$
Y	Y	$-X$	Z
Y	Z	X	Y
Z	$\mathbb{1}$	Z	$\mathbb{1}$
Z	X	Z	X
Z	Y	$\mathbb{1}$	Y
Z	Z	$\mathbb{1}$	Z

Table 1.8: Truth table for CNOT gate employed by the CNF algorithm. C and T refer to control and target qubit, respectively. The primed columns give the values after the operation.

Chapter 2

The Normal Forms [1]

2.1 Row-Reduced Echelon Form

The *Row-Reduced Echelon Form* (RREF) is a normal form using elementary row operations only. In this normal form, the stabilizer state represented by the stabilizer array is not changed, is applicable to states on any number of parties and is an efficient way to eliminate linearly dependent rows from the array. There are three case:

$$\left(\begin{array}{c|c} \mathbb{1} & \\ \vdots & \text{RREF}' \\ \mathbb{1} & \end{array}\right), \quad \left(\begin{array}{c|c} \sigma & * \cdots * \\ \hline \mathbb{1} & \\ \vdots & \text{RREF}' \\ \mathbb{1} & \end{array}\right) \text{ and } \left(\begin{array}{c|c} \sigma_1 & * \cdots * \\ \sigma_2 & * \cdots * \\ \hline \mathbb{1} & \\ \vdots & \text{RREF}' \\ \mathbb{1} & \end{array}\right).$$

RREF$'$ is a sub-array that is also in RREF form. The symbol $*$ denotes either a Pauli operator or an identity $\mathbb{1}$. Furthermore, σ, σ_1 and σ_2 are Pauli operators, where σ_1 and σ_2 anticommute.

The RREF algorithm works by applying a sequence of elementary row operations to the stabilizer array.

Count the number of different Pauli operators in the first column.

a) *If there are no Pauli operators in column 1*, consider that column done.

b) *If there is only 1 kind of Pauli operator*, first make first row with this Pauli the top row, then multiply top row with all other rows with same Pauli in column 1 and consider column 1 and top row done.

c) *If there are at least 2 different kinds of Pauli operators*, let k_1 be the first row where column 1 contains a Pauli operator and k_2 be the first row

where it contains a different Pauli operator. Following that make row k_1 the top row and row k_2 the second row. Then multiply every other row with either the top row, second row, both or none, depending on the element in column 1 as described in Table 2.1. Thereafter continue until all columns/rows done.

Initial stabilizer array: $\begin{pmatrix} \sigma_1 & *\cdots* \\ \sigma_2 & *\cdots* \\ g & *\cdots* \end{pmatrix}$
Depending on the content of row 3, do the following:
$g = \mathbb{1}$: Do nothing
$g = \sigma_1$: Multiply row 1 with row 3.
$g = \sigma_2$: Multiply row 2 with row 3.
$g = \sigma_3$: Multiply row 1 with row 3 and then row 2 with row 3.

Table 2.1: Required operations to eliminate any Pauli operator from row 3 of the stabilizer array. The operators σ_1, σ_2 and σ_3 are a permutation of X, Y and Z.

2.1.1 Checking independence of a set of generators

The easiest way to check independence of a set of generators is to compute the RREF of the stabilizer array. Dependencies between generators will show up as RREF rows containing only $\mathbb{1}$ operators. Removing these all-$\mathbb{1}$ rows leaves an independent set of generators.

2.1.2 Partial Trace of a Stabilizer State

A useful and important operation is the partial trace. The RREF algorithm is the central part in the following efficient partial trace algorithm:

Algorithm PTRACE

1. By column permutations bring the columns of the qubits to be traced out in first position.

2. Bring those columns to RREF.

3. Remove the rows containing the column leader(s).

4. Finally, remove those columns themselves.

Proof. To prove that this algorithm indeed calculates the partial trace, consider once again the three cases for the RREF:

$$\left(\begin{array}{c|c} \mathbb{1} & \\ \vdots & \text{RREF}' \\ \mathbb{1} & \end{array}\right), \quad \left(\begin{array}{c|c} \sigma & * \cdots * \\ \hline \mathbb{1} & \\ \vdots & \text{RREF}' \\ \mathbb{1} & \end{array}\right) \text{ and } \left(\begin{array}{c|c} \sigma_1 & * \cdots * \\ \sigma_2 & * \cdots * \\ \hline \mathbb{1} & \\ \vdots & \text{RREF}' \\ \mathbb{1} & \end{array}\right).$$

We have to show that the state described by RREF', say ρ', is the state obtained from the original stabilizer state ρ by tracing out the qubit pertaining to column 1. Denote the sequences of $*$ operators by g, g_1 and g_2, respectively.

In the first case, $\rho = \frac{\mathbb{1}}{2} \otimes \rho'$ and tracing out qubit 1 yields $Tr_1\rho = \rho'$. In the second case,

$$\rho = \frac{\mathbb{1} \otimes \mathbb{1} + \sigma \otimes g}{2}(\mathbb{1} \otimes \rho') = \frac{1}{2}(\mathbb{1} \otimes \rho' + \sigma \otimes g\rho')$$

and again, as Pauli operators have trace 0, $Tr_1\rho = \rho'$. In the third and final case,

$$\begin{aligned} \rho &= \tfrac{\mathbb{1}\otimes\mathbb{1}+\sigma_1\otimes g_1}{2}\tfrac{\mathbb{1}\otimes\mathbb{1}+\sigma_2\otimes g_2}{2}(\mathbb{1} \otimes 2\rho') \\ &= \tfrac{1}{2}(\mathbb{1} \otimes \rho' + \sigma_1 \otimes g_1\rho' + \sigma_2 \otimes g_2\rho' + \sigma_1\sigma_2 \otimes g_1g_2\rho'), \end{aligned}$$

resulting yet in $Tr_1\rho = \rho'$.

2.2 Single-Party Normal Form

The CNF algorithm applies a sequence of elementary operations to the stabilizer array. After every iteration of the algorithm the stabilizer array has the block structure

$$\left(\begin{array}{cccc|ccc|ccc} X & \mathbb{1} & \cdots & \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} \\ \mathbb{1} & X & \cdots & \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbb{1} & \mathbb{1} & \cdots & X & \mathbb{1} & \cdots & \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} \\ \hline \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} & * & \cdots & * & \mathbb{1} & \cdots & \mathbb{1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} & * & \cdots & * & \mathbb{1} & \cdots & \mathbb{1} \end{array}\right).$$

The block containing the asterisks has not yet been brought to normal form. The columns on the left of the block containing the asterisks, correspond to qubits that are in an eigenstate of the X operator, the columns on its right correspond to qubits in a totally mixed state. The final form, after the completion of the algorithm, is

$$\left(\begin{array}{cccc|ccc} X & \mathbb{1} & \cdots & \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} \\ \mathbb{1} & X & \cdots & \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \mathbb{1} & \mathbb{1} & \cdots & X & \mathbb{1} & \cdots & \mathbb{1} \\ \hline \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} & \mathbb{1} & \cdots & \mathbb{1} \end{array}\right).$$

Hence there is a possibility that the rows of the initial stabilizer array might not be independent.

2.2.1 Algorithm CNF

Count the number of different Pauli operators in the first column.

a) *If there are no Pauli operators in column 1*, then transpose column 1 with the last column and decrease the last column by 1.

b) *If there is only 1 kind of Pauli operator*, let k be the first row where column 1 contains a Pauli operator and make row k the top row by transposing it with the top row. Then apply whatever single-qubit operation on the last column that brings that Pauli operator to an X' multiply the top row with all other rows that have an X in column 1. Thereafter consider the elements of the first row to each of the columns beyond the first one that contains in the first row a Pauli different from X and apply a single-qubit operation to turn it into an X. Following. apply a CNOT operation on the columns which now have an X in the first row, with control column 1. Finally, Increase the top row and first column by 1.

c) *If there are at least 2 different kinds of Pauli operators*, let k_1 be the first row where column 1 contains a Pauli operator and k_2 be the first row where column 1 contains a different Pauli operator. Make row k_1 the top row by transposing row k_1 with the top row. Then make row k_2 the second row by transposing row k_2 with the top row $+1$. Afterwards bring the element on the top row to an X and the element on the top

row +1 to a Z by applying a single-qubit operation on the first column. Thereafter consider the first two rows' find the first column beyond the first one that contains an anti-commuting pair on those rows and bring the anti-commuting pair to an (X,Y) pair by applying a single-qubit operation to that column. Then apply a CNOT operation to that column, with column 1 as control.

d) *If there exists a non-zero size*, continue with step 1, else terminate.

2.2.2 Proof of correctness of algorithm CNF

- In the case where the first column containing $\mathbb{1}$ only, the column is excluded.
- In the case where a column contains $\mathbb{1}$ operators and Pauli operators of just one kind, the first of these Pauli operators is brought to the top row and then the row is excluded. Then a single-qubit rotation is applied to bring the Pauli operators in standard form, which in this case is an X operator.

2.2.3 Alternative proof of projection formulas 1.6 and 1.7

By the proof of the CNF1 algorithm, a state described by a certain stabilizer array is unitarily equivalent to the state described by the normal form of that array.

Let the initial stabilizer group S be given by a stabilizer array. Let S' be the stabilizer group described by the normal form of that array, with K generators

$$g'_k = \mathbb{1} \otimes \ldots \otimes X \otimes \ldots \otimes \mathbb{1},$$

with the X operator in the k-th tensor factor. The stabilizer state corresponding to the normal form is therefore

$$\begin{aligned} \rho' &= \tfrac{1}{2^N} \sum_{i_1 \ldots i_K \in \{0,1\}} X^{i_1} \otimes \ldots \otimes X^{i_K} \otimes \mathbb{1}^{\otimes N-K} \\ &= ((\mathbb{1}+X)/2)^{\otimes K} \otimes (\mathbb{1}/2)^{\otimes N-K} \\ &= \tfrac{1}{2^{N-K}} \prod_{k=1}^{K} \tfrac{\mathbb{1}+g'_k}{2} \end{aligned} .$$

Let U be the unitary corresponding to the sequence of elementary operations that brought stabilizer array to its normal form. S consists of the elements $g = Ug'U^\dagger, g' \in S'$, and can be generated by generators $g_k \equiv Ug'_kU^\dagger$. Then the stabilizer state corresponding to S is given by

$$\begin{aligned}\rho &= \tfrac{1}{2^N}\sum_{g\in S} g = \tfrac{1}{2^N}\sum_{g'\in S'} Ug'U^\dagger = Ug'U^\dagger \\ &= \tfrac{1}{2^{N-K}}\prod_{k=1}^{K} U\tfrac{\mathbb{1}+g'_k}{2}U^\dagger = \tfrac{1}{2^{N-K}}\prod_{k=1}^{K}\tfrac{\mathbb{1}+g_k}{2}.\end{aligned}$$

Chapter 3

Implementation using Maple

3.1 Row-Reduced Echelon Form

In order to produce RREF in maple, first Maple has to make use of a package called *LinearAlgebra* which is a revised package of an older version known as *linalg*. In this spreadsheet, both packages were used as follows,

```
> restart:with(linalg):with(LinearAlgebra):
```

where the command *restart* is used by maple to restart the session and clear the internal memory of the kernel. Following that, Maple has to recognize the Pauli operators X, Y and Z and the identity matrix,

```
> X := Matrix(2,2,[0,1,1,0]):  Y := Matrix(2,2,[0,-I,I,0]):
> Z := Matrix(2,2,[1,0,0,-1]):  II := Matrix(2,2,[1,0,0,1]):
```

Before Maple runs through the algorithm, last thing left to do is assign a random stabilizer array,

```
> RSA := array (1..dim,1..dim, [[X, Z, II, II, II], [Z, X, Z,
Z, II], [II, Z, X, II, Z], [II, Z, II, X, II], [II, II, Z, II,
X]] );
```

where *dim* is the dimension of the stabilizer array which in this case should be assigned number 5 since there are 5 qubits in the stabilizer array assigned into Maple kernel as shown above. In order to check whether the stabilizer sets commute with each other, a simple routine can be constructed shown as Algorithm 1,

If the stabilizer sets commute with each other then the following outcome should be obtained from Maple,

$$i,\ j,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

where i and j indicate the stabilizer sets and the zero matrix indicates that sets i and j do commute. Throughout the algorithm, Maple will have

Algorithm 1 A simple routine which performs the commutation relation in between the stabilizer sets to check whether they commute.

```
> for i from 1 to dim - 1 do
>     for j from i to dim - 1 do
>         Check := 0:
>         for k from 1 to dim do
>             Check := Check + MatrixMatrixMultiply(RSA[i,
k],RSA[j + 1, k]) - MatrixMatrixMultiply(RSA[j + 1,
k],RSA[i, k])
>         od:
>         print( i , j + 1, 'Check' = Check );
>     od:
> od:
```

to recognize whether two matrices are identical using a procedure shown as Algorithm 2,

Algorithm 2 A procedure which enables Maple kernel to identify whether two matrices equal each other.

```
> Equal_Matrix := proc (M1,M2)
>    local Equal_Element, r, c:
>    Equal_Element := 0:
>      for r from 1 to 2 do
>          for c from 1 to 2 do
>              if M1[r, c] = M2[r, c] then Equal_Element :=
Equal_Element + 1
>              fi:
>          od:
>      od:
> end:
```

In order now to obtain the RREF form of the random stabilizer array, Maple has to go through a number of *for* loops. One way of doing so is the following,

Algorithm 3 RREF: Identify the Number of Different Paulis in column n in the active region

```
> NDP := 0:  XP := 0:  YP := 0:  ZP := 0:
>     for m from n to dim do
>         if Equal_Matrix(RSA[m, n], X) = 4 and XP = 0 then
XP := 1
>         elif Equal_Matrix(RSA[m, n], Y) = 4 and YP = 0
then YP := 1
>         elif Equal_Matrix(RSA[m, n], Z) = 4 and ZP = 0
then ZP := 1
>         fi:
>     od:
> NDP := XP + YP + ZP:
```

Algorithm 4 RREF: Identify which row contains the Pauli operator for the case where NDP = 1 and thereafter swap it with the top row in the active region. Its important to note that elementary operations take place throughout the whole stabilizer array.

```
> RS := 0:
>     for l from n to dim while RS = 0 and NDP = 1 do
>         if Equal_Matrix(RSA[l, n], X) = 4 then RS := l
>         elif Equal_Matrix(RSA[l, n], Y) = 4 then RS := l
>         elif Equal_Matrix(RSA[l, n], Z) = 4 then RS := l
>         fi:
>     od:
> if NDP = 1 then RSA := swaprow(RSA, n, RS) fi:
```

Algorithm 5 RREF: Multiply the row which contains the same Pauli operator as the one in the top row.

```
>  for i from n to dim - 1 while NDP = 1 do
>      if Equal_Matrix(RSA[n, n], X) = 4 and
Equal_Matrix(RSA[i + 1, n], X) = 4 then
>          for j from 1 to dim do
>              RSA[i + 1, j] := MatrixMatrixMultiply(RSA[n,
j], RSA[i + 1, j])
>          od:
>      elif Equal_Matrix(RSA[n, n], Y) = 4 and
Equal_Matrix(RSA[i + 1, n], Y) = 4 then
>          for j from 1 to dim do
>              RSA[i + 1, j] := MatrixMatrixMultiply(RSA[n,
j], RSA[i + 1, j])
>          od:
>      elif Equal_Matrix(RSA[n, n], Z) = 4 and
Equal_Matrix(RSA[i + 1, n], Z) = 4 then
>          for j from 1 to dim do
>              RSA[i + 1, j] := MatrixMatrixMultiply(RSA[n,
j], RSA[i + 1, j])
>          od:
>      fi:
>  od:
```

Algorithm 6 RREF: As in Algorithm 3, this algorithm identifies how many different Pauli operators are in column n and assigns the row number for each different Pauli.

```
>  XP := 0:  YP := 0:  ZP := 0:  NDP := 0:  XRS := 0:  YRS
:= 0:  ZRS := 0:
>      for m from n to dim do
>          if Equal_Matrix(RSA[m, n], X) = 4 and XP = 0 then
>             XP := 1:
>             XRS := m:
>          elif Equal_Matrix(RSA[m, n], Y) = 4 and YP = 0
then
>             YP := 1:
>             YRS := m:
>          elif Equal_Matrix(RSA[m, n], Z) = 4 and ZP = 0
then
>             ZP := 1:
>             ZRS := m:
>          fi:
>      od:
>  NDP := XP + YP + ZP:
```

Algorithm 7 RREF: Swap the top two rows of the stabilizer array with the first two rows which contain different Pauli operators.

```
> Stop_X := 0:  Stop_Y := 0:  Stop_Z := 0:  KU := n:
>     for k from n to dim while NDP > 1 do
>         if Equal_Matrix(RSA[k, n], X) = 4 and Stop_X = 0
then
>            RSA := swaprow(RSA, KU, XRS):
>            Stop_X := 1:
>            KU := KU + 1:
>         elif Equal_Matrix(RSA[k, n], Y) = 4 and Stop_Y =
0 then
>            RSA := swaprow(RSA, KU, YRS):
>            Stop_Y := 1:
>            KU := KU + 1:
>         elif Equal_Matrix(RSA[k, n], Z) = 4 and Stop_Z =
0 then
>            RSA := swaprow(RSA, KU, ZRS):
>            Stop_Z := 1:
>            KU := KU + 1:
>         fi:
> od:
```

Algorithm 8 RREF: Perform Pauli elimination operations according to Table 2.1.

```
> for p from n to dim - 2 while NDP > 1 do
>     if Equal_Matrix(RSA[p + 2, n], II) = 4 then next
>     elif Equal_Matrix(RSA[n, n], RSA[p + 2, n]) = 4 then
>        for j from 1 to dim do
>            RSA[p + 2, j] := MatrixMatrixMultiply(RSA[n,
j], RSA[p + 2, j])
>        od:
>     elif Equal_Matrix(RSA[n + 1, n], RSA[p + 2, n]) = 4
then
>        for j from 1 to dim do
>            RSA[p + 2, j] := MatrixMatrixMultiply(RSA[n +
1, j], RSA[p + 2, j])
>        od:
>     else
>        for j from 1 to dim do
>            RSA[p + 2, j] := MatrixMatrixMultiply(RSA[n,
j], RSA[p + 2, j]):
>            RSA[p + 2, j] := MatrixMatrixMultiply(RSA[n +
1, j], RSA[p + 2, j]):
>        od:
>     fi:
> od:
```

Algorithms 3 to 8 should be enclosed in a *for* loop,

```
> for n from 1 to dim - 1 do
>     Algorithm 3
>     .
>     .
>     .
>     Algorithm 8
> od:
```

so that they will be repeated $dim - 1$ times in order to proceed to the next column and row of the stabilizer array where a new active region will be defined. After running through algorithms 3 to 8, $dim - 1$ times, a second check should be performed in order to establish whether the stabilizer sets commute with each other. Refer to Appendix B for the outcome of the algorithm.

3.2 Clifford Normal Form

As it has it has been describe in Chapter 3.1, in order to produce RREF in maple, first Maple has to make use of a package called *LinearAlgebra* which is a revised package of an older version known as *linalg*. The same implies for the CNF as we will be making use of matrix multiplication commands and other commands used by those packages. In this spreadsheet, both packages were used as follows,

```
> restart:with(linalg):with(LinearAlgebra):
```

where the command *restart* is used by maple to restart the session and clear the internal memory of the kernel. Following that, Maple has to recognize the Pauli operators X, Y and Z and the identity matrix,

```
> X := Matrix(2,2,[0,1,1,0]):  Y := Matrix(2,2,[0,-I,I,0]):
> Z := Matrix(2,2,[1,0,0,-1]):  II := Matrix(2,2,[1,0,0,1]):
```

Before Maple runs through the individual algorithms, last thing left to do is assign a random stabilizer array,

```
> RSA := array (1..K,1..N, [[X, II, II, II, II], [II, II, Z,
II, Y], [II, II, Y, Y, Z], [II, II, X, II, X]] );
```

where K and N are the dimensions of the stabilizer array which in this case should be assigned number 4 and number 5, respectively since there are 4 stabilizer sets and 5 qubits in the stabilizer array assigned into Maple kernel as shown above.

Once again Algorithm 2 will be used so that Maple will recognize whether two matrices are identical.

In order now to obtain the CNF form of the random stabilizer array, Maple has to go through a number of Algorithms individually. The decision,

which Algorithm will be used first and which will follow, is based entirely on the description of the CNF Algorithm in Chapter 2.2.1. In order to measure the number of different Pauli operators, we make use of Algorithm 3 and assign *dim* with the number of stabilizer sets, which in this case is four. For the first case where there are no Pauli operators in the first column of the active region, we make use of the command,

```
> RSA := swapcol(RSA, NL, NR):
> NR := NR - 1:
```

to swap columns N_L(the first column in the active region) and N_R(the last column in the active region) in the stabilizer array and then decrease N_R by 1.

For the second case we make use of the command,

```
> RSA := swaprow(RSA, KU, RS):
```

to swap rows K_U(the first row in the active region) and RS, which must be assigned its value using Algorithm 4. Using Table 1.7, we apply any single-qubit operation on the first column of the active region, N_L, so that the Pauli operator in column N_L and row K_U will be transformed to an X. Using the following Algorithm,

```
> for k from KU to K - 1 do
>     if Equal_Matrix(RSA[KU, NL], X) = 4 and Equal_Matrix(RSA[k
+ 1, NL], X) = 4 then
>         RSA[k + 1, NL] := MatrixMatrixMultiply(RSA[KU, NL],
RSA[k + 1, NL]):
>     fi:
> od:
```

which is a shorter version of Algorithm 5, we multiply K_U with all other rows in the active region that have X in column N_L. Considering the elements of the first row of the active region, apply a single-qubit operation to each of the columns beyond the first one, to turn the Pauli which is different from X, into an X, by making use of Algorithm 9,

Considering now the same columns, which now have an X in the first row, apply a CNOT operation with control column N_L as is shown in Table 1.8, using Algorithm 10. After this operation, K_U and N_L must increase by 1 in order to decrease the size of the active region.

For the third case, Algorithms 6 and 7 must be used in order to make row K_U the row with the first different Pauli operator and row $K_U + 1$ the row with the second different Pauli operator. Thereafter Algorithm 9 should be used once again as the single-qubit operation. When considering the first two rows of the active region, K_U and K_U+1, then the single-qubit operation Algorithm must be amended as shown in Algorithm 11, since the operation takes place throughout the rows.

Algorithm 9 CNF: Single-qubit operation.

```
>  for n from NL + 1 to N do
>      if Equal_Matrix(RSA[KU, n], Y) = 4 and
Equal_Matrix(RSA[KU + 1, n], Z) = 4 then
>          RSA[KU, n] := X: RSA[KU + 1, n] := - Z:
>          for k from KU + 2 to K while Equal_Matrix(RSA[k,
n], X) = 4 do
>              RSA[k, n] := Y
>          od:
>      elif Equal_Matrix(RSA[KU, n], X) = 4 and
Equal_Matrix(RSA[KU + 1, n], Y) = 4 then
>          RSA[KU, n] := - X: RSA[KU + 1, n] := Z:
>          for k from KU + 2 to K while Equal_Matrix(RSA[k,
n], Z) = 4 do
>              RSA[k, n] := Y
>          od:
>      elif Equal_Matrix(RSA[KU, n], Y) = 4 and
Equal_Matrix(RSA[KU + 1, n], X) = 4 then
>          RSA[KU, n] := X: RSA[KU + 1, n] := Z:
>          for k from KU + 2 to K while Equal_Matrix(RSA[k,
n], Z) = 4 do
>              RSA[k, n] := Y
>          od:
>      elif Equal_Matrix(RSA[KU, n], Z) = 4 and
Equal_Matrix(RSA[KU + 1, n], X) = 4 then
>          RSA[KU, n] := X: RSA[KU + 1, n] := Z:
>          for k from KU + 2 to K while Equal_Matrix(RSA[k,
n], Y) = 4 do
>              RSA[k, n] := - Y
>          od:
>      elif Equal_Matrix(RSA[KU, n], Z) = 4 and
Equal_Matrix(RSA[KU + 1, n], Y) = 4 then
>          RSA[KU, n] := X: RSA[KU + 1, n] := Z:
>          for k from KU + 2 to K while Equal_Matrix(RSA[k,
n], X) = 4 do
>              RSA[k, n] := Y
>          od:
>      fi:
>  od:
```

Algorithm 10 CNF: CNOT operation.

```
>  for n from NL + 1 to N do
>      for k from KU to K do
>          for j from 1 to N - n do
>              if Equal_Matrix(RSA[k, n], II) = 4 and
Equal_Matrix(RSA[k, n + j], II) = 4 then next
>              elif Equal_Matrix(RSA[k, n], II) = 4 and
Equal_Matrix(RSA[k, n + j], X) = 4 then next
>              elif Equal_Matrix(RSA[k, n], II) = 4 and
Equal_Matrix(RSA[k, n + j], Y) = 4 then
>                  RSA[k ,n] := Z: RSA[k ,n + j] := Y:
>              elif Equal_Matrix(RSA[k, n], II) = 4 and
Equal_Matrix(RSA[k, n + j], Z) = 4 then
>                  RSA[k ,n] := Z: RSA[k ,n + j] := Z:
>              elif Equal_Matrix(RSA[k, n], X) = 4 and
Equal_Matrix(RSA[k, n + j], II) = 4 then
>                  RSA[k ,n] := X: RSA[k ,n + j] := X:
>              elif Equal_Matrix(RSA[k, n], X) = 4 and
Equal_Matrix(RSA[k, n + j], X) = 4 then
>                  RSA[k ,n] := X: RSA[k ,n + j] := II:
>              elif Equal_Matrix(RSA[k, n], X) = 4 and
Equal_Matrix(RSA[k, n + j], Y) = 4 then
>                  RSA[k ,n] := Y: RSA[k ,n + j] := Z:
>              elif Equal_Matrix(RSA[k, n], X) = 4 and
Equal_Matrix(RSA[k, n + j], Z) = 4 then
>                  RSA[k ,n] := - Y: RSA[k ,n + j] := Y:
>              elif Equal_Matrix(RSA[k, n], Y) = 4 and
Equal_Matrix(RSA[k, n + j], II) = 4 then
>                  RSA[k ,n] := Y: RSA[k ,n + j] := X:
>              elif Equal_Matrix(RSA[k, n], Y) = 4 and
Equal_Matrix(RSA[k, n + j], X) = 4 then
>                  RSA[k ,n] := Y: RSA[k ,n + j] := II:
>              elif Equal_Matrix(RSA[k, n], Y) = 4 and
Equal_Matrix(RSA[k, n + j], Y) = 4 then
>                  RSA[k ,n] := - X: RSA[k ,n + j] := Z:
>              elif Equal_Matrix(RSA[k, n], Y) = 4 and
Equal_Matrix(RSA[k, n + j], Z) = 4 then
>                  RSA[k ,n] := X: RSA[k ,n + j] := Y:
>              elif Equal_Matrix(RSA[k, n], Z) = 4 and
Equal_Matrix(RSA[k, n + j], II) = 4 then next
>              elif Equal_Matrix(RSA[k, n], Z) = 4 and
Equal_Matrix(RSA[k, n + j], X) = 4 then
>                  RSA[k ,n] := Z: RSA[k ,n + j] := X:
>              elif Equal_Matrix(RSA[k, n], Z) = 4 and
Equal_Matrix(RSA[k, n + j], Y) = 4 then
>                  RSA[k ,n] := II: RSA[k ,n + j] := Y:
>              elif Equal_Matrix(RSA[k, n], Z) = 4 and
Equal_Matrix(RSA[k, n + j], Z) = 4 then
>                  RSA[k ,n] := II: RSA[k ,n + j] := Z:
>              fi:
>          od:
>      od:
>  od:
```

Algorithm 11 CNF: Single-qubit operation revised.

```
>  for n from NL + 1 to N do
>      if Equal_Matrix(RSA[KU, n], Y) = 4 and
Equal_Matrix(RSA[KU + 1, n], Z) = 4 then
>          RSA[KU, n] := X: RSA[KU + 1, n] := Y:
>          for k from KU + 2 to K while Equal_Matrix(RSA[k,
n], X) = 4 do
>              RSA[k, n] := Z
>          od:
>      elif Equal_Matrix(RSA[KU, n], Y) = 4 and
Equal_Matrix(RSA[KU + 1, n], X) = 4 then
>          RSA[KU, n] := X: RSA[KU + 1, n] := Y:
>          for k from KU + 2 to K while Equal_Matrix(RSA[k,
n], Z) = 4 do
>              RSA[k, n] := - Z
>          od:
>      elif Equal_Matrix(RSA[KU, n], Z) = 4 and
Equal_Matrix(RSA[KU + 1, n], X) = 4 then
>          RSA[KU, n] := X: RSA[KU + 1, n] := Y:
>          for k from KU + 2 to K while Equal_Matrix(RSA[k,
n], Y) = 4 do
>              RSA[k, n] := Z
>          od:
>      elif Equal_Matrix(RSA[KU, n], Z) = 4 and
Equal_Matrix(RSA[KU + 1, n], Y) = 4 then
>          RSA[KU, n] := X: RSA[KU + 1, n] := - Y:
>          for k from KU + 2 to K while Equal_Matrix(RSA[k,
n], X) = 4 do
>              RSA[k, n] := Z
>          od:
>      elif Equal_Matrix(RSA[KU, n], X) = 4 and
Equal_Matrix(RSA[KU + 1, n], Z) = 4 then
>          RSA[KU, n] := - X: RSA[KU + 1, n] := Y:
>          for k from KU + 2 to K while Equal_Matrix(RSA[k,
n], Y) = 4 do
>              RSA[k, n] := Z
>          od:
>      fi:
>  od:
```

Finally apply Algorithm 10 and terminate the operation when the active region has zero-size, else repeat the procedure.

Conclusion

Stabilizer codes are with out any doubt the most convenient way of representing long winded states and thus they can be regarded as the future of quantum information and quantum computing.

Throughout this review of [1], I showed that the RREF and CNF forms can be implemented in another mathematical package apart from MatLab as it has been already stated by the writers of [1]. My main goal was to implement the CNF algorithm in Maple the way I implemented the RREF algorithm, i.e. compute the reduced forms by running the stabilizer array through a single algorithm and not by individually running the stabilizer array through smaller algorithms. I admit that even the RREF algorithm had not been thoroughly checked and also I have mot crossed referenced my results with the MatLab routines, provided by the writers of [1]. This was due to the lack of time and also due to my low knowledge in the MatLab package. Also as I have realized throughout this dissertation, it is absolutely tedious to try and implement an algorithm which will not stack in any random stabilizer array, as there are many different examples and therefore it is much easier to run the stabilizer array through much smaller algorithms individually according to the constrains of the RREF/CNF algorithms.

Acknowledgments

The academic year 2008/09 was probably the worst year of my career as a physicist. This is due to my failure to pass the exams in the Master of Science for which this dissertation was submitted as a partial fulfillment of the requirements. Nevertheless I have enjoyed the last few months more than anything as i got great satisfaction from successfully producing a dissertation in spite of the psychologically difficult period I endured. The successful completion of this dissertation was all I needed to regain my will to move on with my career as a physicist. Of course this would not have been possible if my supervisor, Dr Terry G Rudolph had not given me the chance to work with him despite my failure to pass his exam.

Also I would like to thank my friend, Cecilia N Gyansah who has supported me this year and encouraged me to move from this challenging year. I wish her the best in her medical career and that she becomes one of the greatest doctors.

Bibliography

[1] Koenraad M.R. Audenaert and Martin B. Plenio, E-print arXiv quant-ph/0505036.

[2] M.A. Nielsen and I.L Chuang, *Quantum computation and quantum information*, Cambridge University Press, 2000.

[3] Daniel Gottesman, *Stabilizer Codes and Quantum Error Correction*, quant-ph/9705052v1

[4] Jaeger G. Quantum information. An overview (Springer, 2007)(ISBN 0387357254)(291s)CsQc

[5] Nakahara9789812814487

Appendix A

Group Theory

In this appendix some of the fundamental concepts and important definitions are summarized.

A.1 Basic definitions

A group $(G, \cdot)$ is a non-empty set G with a binary group multiplication operation '$\cdot$', with the following properties:

- *closure*, $g_1 \cdot g_2 \in G$ for all $g_1, g_2 \in G$.
- *associativity*, $(g_1 \cdot g_2) \cdot g_3 = g_1 \cdot (g_2 \cdot g_3)$, for all $g_1, g_2, g_3 \in G$.
- *identity*, $\exists e \in G$ such that $\forall g \in G, g \cdot e = e \cdot g = g$.
- *inverses*, $\forall g \in G, \exists g^{-1} \in G$ such that $g \cdot g^{-1} = e$ and $g^{-1} \cdot g = e$.

A group G is *finite* if the number of elements in G is finite. The *order* of a finite group G is the number of elements it contains, denoted as $|G|$. A group G is said to be *abelian* if $g_1 g_2 = g_2 g_1 \forall g_1, g_2 \in G$. The *order* of an element $g \in G$ is the smallest positive integer r such that g^r equals the identity element e. A *subgroup* H of G is a subset of G which forms a group under the same group multiplication operation as G.

Theorem A.1.1 (Lagrange's Theorem) *Let G be a finite group, and let H be a subgroup of G. Then the order of H divides the order of G.*

If g_1 and g_2 are elements of G, then the conjugate of g_2 with respect to g_1 is the element $g_1^{-1} g_2 g_1$. If H is subgroup of G, then it is known as a normal subgroup if $g^{-1} H g = H \forall g \in G$. The *conjugacy class* G_x of an element x in a group G is defined by $G_x \equiv \{g^{-1} x g | g \in G\}$.

The *Pauli group* on n qubits is a non-Abelian group. For a single qubit, the Pauli group is defined to consist of all the Pauli matrices with multiplicative factors $\pm\mathbb{1}$, $\pm i$,

$$G_1 \equiv \{\pm\mathbb{1}, \pm i\mathbb{1}, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\}\,.$$

This set of matrices forms a group under the operation of matrix multiplication.

A.2 Generators

The study of groups is often greatly simplified by the use of a set of *group generators* for the group being studied. A set of elements $g_1...g_l$ in a group G is said to *generate* the group G if every element of G can be written as a product of elements from the list $g_1...g_l$ and $G = \langle g_1...g_l\rangle$.

The great advantage of using generators to describe groups is that they provide a *compact* way of describing the group. Suppose G has size $|G|$ and $g_1...g_l$ is a set of elements in group G and g is not an element of $\langle g_1...g_l\rangle$. Let $f \in \langle g_1...g_l\rangle$. Then $fg \notin \langle g_1...g_l\rangle$ since if it were then we would have $g = f^{-1}fg \in \langle g_1...g_l\rangle$ which is false. Thus for each element $f \in \langle g_1...g_l\rangle$ $\exists fg \in \langle g_1...g_l, g\rangle$ but $fg \notin \langle g_1...g_l\rangle$. Thus adding the generator g to $\langle g_1...g_l\rangle$ increases the size of the group being generated, from which we conclude that G must have a set of generators containing at most $log\,(|G|)$ elements.

A.3 Cyclic groups

A *cyclic group* G possesses an element a such that any element $g \in G$ can be expressed as a^n for some integer n. a is known as a *generator* of G and $G = \langle a\rangle$. A *cyclic subgroup* H generated by $g \in G$ is the group formed by $\{e, g, g^2...g^{r-1}\}$, where r is the order of g and therefore, $H = \langle g\rangle$.

Appendix B

RREF: Maple Outcome

Here are shown the results provided by Maple after going through the implementation of RREF form in Maple as described in section 3.1. First the random stabilizer array takes the following form as presented by Maple,

$$RSA := \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{bmatrix}$$

After going through the commutation relation algorithm 1, to check whether the stabilizer sets commute with one another, the following output should be presented,

$$1,\, 2,\, Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$1,\, 3,\, Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$1,\ 4,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$1,\ 5,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$2,\ 3,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$2,\ 4,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$2,\ 5,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$3,\ 4,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$3,\ 5,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$4,\ 5,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

After going through algorithms 3 to 8, the following outcome is produced with intermediate steps which are presented on request, in order to produce the final result,

$$\texttt{`LinearAlgebra:-Column`} = 1$$

$$\textit{ElementaryOperation1} - \textit{RowSwapKU}$$

$$\begin{bmatrix}
\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}
\end{bmatrix}$$

$$\textit{ElementaryOperation2} - \textit{RowSwapKU} + 1$$

$$\begin{bmatrix}
\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}
\end{bmatrix}$$

‘LinearAlgebra:-Column‘ = 2

ElementaryOperation1 − RowSwapKU

$$\begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{bmatrix}$$

ElementaryOperation2 − RowSwapKU + 1

$$\begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{bmatrix}$$

ElementaryOperation3 $-$ *KU* $+$ *1_SamePauliOperatorMatrixMultiply*

$$\begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{bmatrix}$$

‘LinearAlgebra:-Column‘ $= 3$

ElementaryOperation1 $-$ *RowSwapKU*

$$\begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{bmatrix}$$

$$ElementaryOperation2 - RowSwapKU + 1$$

$$\begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \end{bmatrix}$$

$$ElementaryOperation3 - KU\_SamePauliOperatorMatrixMultiply$$

$$\begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix}$$

$$\texttt{`LinearAlgebra:-Column`} = 4$$

$$ElementaryOperation1 - RowSwapKU$$

$$\begin{bmatrix}
\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}
\end{bmatrix}$$

And final going through the commutation relation algorithm 1 the following outcome should be received, proving the legitimacy of the stabilizer array in its RREF form,

$$1,\ 2,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$1,\ 3,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$1,\ 4,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$1,\ 5,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$2,\ 3,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$2,\ 4,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$2,\ 5,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$3,\ 4,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$3,\ 5,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$4,\ 5,\ Check = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Appendix C

CNF Example [1]

Initial Random Stabilizer Array(RSA),

$$RSA := \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{bmatrix}$$

Swap second column with the rightmost column NR and decrease NR by 1,

$$RSA := \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix}$$

Perform single-qubit operation on the second column which is the leftmost column of the active region known as NL,

$$RSA := \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix}$$

Perform single-qubit operation on the third column of the stabilizer array or the second column of the active region,

$$RSA := \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix}$$

Perform a CNOT operation on the second column of the active region with control column NL,

$$RSA := \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix}$$

Multiply the last row of the active region with its first row,

$$RSA := \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix}$$

Perform a single-qubit operation on the third column of the stabilizer array, since KU and NL have been increased by 1,

$$RSA := \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix}$$

Multiply the fourth row of the stabilizer array with the third row,

$$RSA := \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix}$$

Perform a single-qubit operation on the fourth column of the stabilizer array,

$$RSA := \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix}$$

Finally, perform a CNOT operation on the fourth column of the stabilizer array,

$$RSA := \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix}$$