Chapter 2

Saving qubits with classical codes

2.1 Background

One essential component in realizing simulations of fermionic models on quantum computers is the representation of such models in terms of qubits and quantum gates. Following initial simulation schemes for fermions hopping on a lattice [3], more recent proposals used the Jordan-Wigner [19] transform [21–24], the Verstraete-Cirac mapping [25], or the Bravyi-Kitaev transform [26] to find a suitable representation. Specifically, the task of all such representations is two-fold. First, we seek a mapping from states in the fermionic Fock space of N sites to the space of n qubits. The fermionic Fock space is spanned by 2^N basis vectors $|\nu_1, \ldots, \nu_N\rangle$ where $\nu_j \in \{0, 1\}$ indicates the presence ($\nu_j = 1$) or absence ($\nu_j = 0$) of a spinless fermionic particle at orbital j. Such a mapping $\mathbf{e} : \mathbb{Z}_2^{\otimes N} \mapsto \mathbb{Z}_2^{\otimes n}$ is also called an *encoding* [27]. An example of such an encoding is the trivial one in which $n = N$ and qubits are used to represent the binary string $\boldsymbol{\nu} = (\nu_1, \ldots, \nu_N)^\top$. That is,

$$|\boldsymbol{\omega}\rangle = |\mathbf{e}(\boldsymbol{\nu})\rangle = \bigotimes_{j=1}^{n} |\omega_j\rangle \ , \tag{2.1}$$

where $\omega_j = \nu_j$ in the standard basis $\{|0\rangle, |1\rangle\}$.

Second, we need a way to simulate the dynamics of fermions on these N orbitals. These dynamics can be modeled entirely in terms of

the annihilation and creation operators c_j and $c_j^\dagger$ that satisfy the anti-commutation relations (1.7). Following these relations, the operators act on the fermionic Fock space as

$$c_{i_m}^\dagger \, c_{i_1}^\dagger ... \, c_{i_{m-1}}^\dagger c_{i_m}^\dagger c_{i_{m+1}}^\dagger ... \, c_{i_M}^\dagger \, |\Theta\rangle = 0 \tag{2.2}$$

$$c_{i_m} \, c_{i_1}^\dagger ... \, c_{i_{m-1}}^\dagger c_{i_{m+1}}^\dagger ... \, c_{i_M}^\dagger \, |\Theta\rangle = 0 \tag{2.3}$$

$$\begin{aligned} & c_{i_m} \, c_{i_1}^\dagger ... \, c_{i_{m-1}}^\dagger c_{i_m}^\dagger c_{i_{m+1}}^\dagger ... \, c_{i_M}^\dagger \, |\Theta\rangle \\ = (-1)^{m-1} \, & c_{i_1}^\dagger ... \, c_{i_{m-1}}^\dagger c_{i_{m+1}}^\dagger ... \, c_{i_M}^\dagger \, |\Theta\rangle \end{aligned} \tag{2.4}$$

$$\begin{aligned} & c_{i_m}^\dagger \, c_{i_1}^\dagger ... \, c_{i_{m-1}}^\dagger c_{i_{m+1}}^\dagger ... \, c_{i_M}^\dagger \, |\Theta\rangle \\ = (-1)^{m-1} \, & c_{i_1}^\dagger ... \, c_{i_{m-1}}^\dagger c_{i_m}^\dagger c_{i_{m+1}}^\dagger ... \, c_{i_M}^\dagger \, |\Theta\rangle \, , \end{aligned} \tag{2.5}$$

where $|\Theta\rangle$ is the fermionic vacuum and $\{i_1, \, ... \, , \, i_M\} \subseteq \{1, \, ... \, , \, N\}$. Mappings of the operators c_j to qubits typically use the Pauli matrices X, Z, and Y acting on one qubit, characterized by their anti-commutation relations $[P_i, P_j]_+ \; = \; 2\,\delta_{ij}\,\mathbb{I}$, for all $P_i \in \mathcal{P} = \{X, Y, Z\}$. An example of such a mapping is the Jordan-Wigner transform [19] given by

$$c_j \mathrel{\hat{=}} Z^{\otimes j-1} \otimes \sigma^- \otimes \mathbb{I}^{\otimes n-j} \tag{2.6}$$

$$c_j^\dagger \mathrel{\hat{=}} Z^{\otimes j-1} \otimes \sigma^+ \otimes \mathbb{I}^{\otimes n-j} \tag{2.7}$$

where

$$\sigma^- \; = \; |0\rangle\langle 1| = \frac{1}{2} \left(X + iY \right) \, , \tag{2.8}$$

$$\sigma^+ \; = \; |1\rangle\langle 0| = \frac{1}{2} \left(X - iY \right) \, . \tag{2.9}$$

It is easily verified that together with the trivial encoding (2.1) this transformation satisfies the desired properties (2.2)-(2.5) and can hence be used to represent fermionic models with qubit systems.

In order to assess the suitability of an encoding scheme for the simulation of fermionic models on a quantum computer, a number of parameters are of interest. The first is the total number of qubits n needed in the simulation. Second, we may care about the gate size of the operators c_j and $c_j^\dagger$ when mapped to qubits. In its simplest form, this problem concerns the total number of qubits on which these operators do not

act trivially, that is, the number of qubits L, on which an operator acts as $P_j \in \mathcal{P}$ instead of the identity $\mathbb{I}$, sometimes called the Pauli length. Different transformations can lead to dramatically different performance with respect to these parameters. For both the Jordan-Wigner as well as the Bravyi-Kitaev transform $n = N$, but we have $L = O(n)$ for the first, while $L = O(\log n)$ for the second. We remark that in experimental implementations we typically do not only care about the absolute number L, but rather the specific gate size and individual difficulty of the qubit gates each of which may be easier or harder to realize in a specific experimental architecture. For error-corrected quantum simulation, the cost in T-gates is as important to optimize as the circuit depth [28], and quantum devices with restricted connectivity even require mappings tailored to them [29, 30]. Finally, we remark that instead of looking for a mapping for individual operators $c_j^{(\dagger)}$ we may instead opt to map pairs (or higher order terms) of such operators at once, or even look to represent sums of such operators.

2.2 Results

Here, we propose a general family of mappings of fermionic models to qubit systems and quantum gates that allow us to trade off the necessary number of qubits n against the difficulty of implementation as parametrized by L, or more complicated quantum gates such as CPHASE. Ideally, one would of course like both the number of qubits, as well as the gate size to be small. We show that our mappings can lead to significant savings in qubits for a variety of examples (see Table 2.1) as compared to the Jordan-Wigner transform for instance, at the expense of greater complexity in realizing the required gates. The latter may lead to an increased time required for the simulation depending on which gates are easy to realize in a particular quantum computing architecture.

At the heart of our efforts is an entirely general construction of the creation and annihilation operators in (2.2) given an arbitrary encoding $\mathbf{e}$ and the corresponding decoding $\mathbf{d}$. As one might expect, this construction is not efficient for every choice of encoding $\mathbf{e}$ or decoding $\mathbf{d}$. However, for linear encodings $\mathbf{e}$, but possibly nonlinear decodings $\mathbf{d}$, they can take on a very nice form. While in principle any classical code with the same properties can be shown to yield such mappings, we provide an appealing example of how a classical code of fixed Hamming weight [31]

can be used to give an interesting mapping.

Two other approaches allow us to be more modest with the algorithmic depth in either accepting a qubit saving that is linear with N, or just saving a fixed amount of qubits for hardly any cost at all.
In previous works, trading quantum resources has been addressed for general algorithms [32], and quantum simulations [33–35]. In the two works of Moll et al. and Bravyi et al., qubit requirements are reduced with a scheme that is different from ours. A qubit Hamiltonian is first obtained with e.g. the Jordan-Wigner transform, then unitary operations are applied to it in order taper qubits off successively. The paper by Moll et al. provides a straightforward method to calculate the Hamiltonian, that can be used to reduce the amount of qubits to a minimum, but the number of Hamiltonian terms scales exponentially with the particle number. The notion that our work is based on, was first introduced in [34] by Bravyi et al., for linear en- and decodings. With the generalization of this method, we hope to make the goal of qubit reduction more attainable in reducing the effort to do so. The reduction method is mediated by nonlinear codes, of which we provide different types to choose from. The transform of the Hamiltonian is straight-forward from there on, and we give explicit recipes for arbitrary codes. We can summarize our contributions as follows.

- We show that for any encoding $\mathbf{e} : \mathbb{Z}_2^{\otimes N} \mapsto \mathbb{Z}_2^{\otimes n}$ there exists a mapping of fermionic models to quantum gates. For the special case that this encoding is linear, our procedure can be understood as a slightly modified version of the perspective taken in [27]. This gives a systematic way to employ classical codes for obtaining such mappings.

- Using particle-conservation symmetry, we develop 3 types of codes that save a constant, linear and exponential amount of qubits (see Table 2.1 and Sections 2.4.3.1-2.4.3.3). An example from classical coding theory [31] is used to obtain significant qubit savings (here called the binary addressing code), at the expense of increased gate difficulty (unless the architecture would easily support multi-controlled gates).

- The codes developed are demonstrated on two examples from quantum chemistry and physics.

- The Hamiltonian of the well-studied hydrogen molecule in minimal basis is re-shaped into a two-qubit problem, using a simple code.
- A Fermi-Hubbard model on a 2×5 lattice and periodic boundary conditions in the lateral direction is considered. We parametrize and compare the sizes of the resulting Hamiltonians, as we employ different codes to save various amounts of qubits. In this way, the trade-off between qubit savings and gate complexity is illustrated (see Table 2.2).

Mapping	En-/Decoding type	Qubits saved	$n(N,K)$	Resulting gates	Origin
Jordan-Wigner | Parity tr.	linear/linear	none	N	length-$O(n)$ Pauli strings	[19, 27]
Bravyi-Kitaev transform	linear/linear	none	N	length-$O(\log n)$ Pauli strings	[26]
Checksum codes	linear/ affine linear	$O(1)$	$N-1$	length-$O(n)$ Pauli strings	[36]
Binary addressing codes	nonlinear/nonlinear	$O(2^{n/K})$	$\log\left(N^K/K!\right)$	$(O(n))$-controlled gates	[36]
Segment codes	linear/nonlinear	$O\left(n/K\right)$	$N/(1+\frac{1}{2K})$	$(O(K))$-controlled gates	[36]

Table 2.1. Overview of mappings presented in this paper, listed by the complexity of their code functions, their qubit savings, qubit requirements (n), properties of the resulting gates and first appearance. Mappings can be compared with respect to the size of plain words (N) and their targeted Hamming weight K. We also refer to different methods that are not listed, as they do not rely on codes in any way [33, 34].

2.3 Encoding the entire Fock space

To illustrate the general use of (possibly nonlinear) encodings to represent fermionic models, let us first briefly generalize how existing mappings can be phrased in terms of linear encodings in the spirit of [27]. Under consideration in representing the dynamics is a mapping for second-quantized Hamiltonians of the form

$$\begin{aligned} H &= \sum_{l=0}^{\infty} \sum_{\substack{\boldsymbol{a}\in[N]^{\otimes l} \\ \boldsymbol{b}\in\mathbb{Z}_2^{\otimes l}}} h_{\boldsymbol{ab}} \prod_{i=1}^{l} (c_{a_i}^{\dagger})^{b_i} (c_{a_i})^{1+b_i \bmod 2} \\ &= \sum_{l} \sum_{\substack{\boldsymbol{a},\boldsymbol{b} \\ \text{with } h_{\boldsymbol{ab}}\neq 0}} \widehat{h}_{\boldsymbol{ab}}\,, \end{aligned} \tag{2.10}$$

where $h_{\boldsymbol{ab}}$ are complex coefficients, chosen in a way as to render H hermitian. For our convenience, we use length-l N-ary vectors $\boldsymbol{a} = (a_1, \ldots, a_l)^{\top} \in [N]^{\otimes l}$ to parametrize the orbitals on which a term $\widehat{h}_{\boldsymbol{ab}}$ is acting, and write $[N] = \{1, \ldots, N\}$. A similar notation will be employed for binary vectors of length l, with $\boldsymbol{b} = (b_1, \ldots, b_l)^{\top} \in \mathbb{Z}_2^{\otimes l}$, $\mathbb{Z}_2 = \{0, 1\}$, deciding whether an operator is a creator or annihilator by the rules $(c_i^{(\dagger)})^1 = c_i^{(\dagger)}$ and $(c_i^{(\dagger)})^0 = 1$.
Every term $\widehat{h}_{\boldsymbol{ab}}$ is a linear operation $\mathcal{F}_N \mapsto \mathcal{F}_N$, with $\mathcal{F}_N$ being the Fock space restricted on N orbitals, the direct sum of all possible anti-symmetrized M-particle Hilbert spaces $\mathcal{H}_N^M$: $\mathcal{F}_N = \bigoplus_{m=0}^{N} \mathcal{H}_N^m$. Conventional mappings transform states of the Fock space $\mathcal{F}_N$ into states on N qubits, carrying over all linear operations as well $\mathcal{L}(\mathcal{F}_N) \mapsto \mathcal{L}((\mathbb{C}^2)^{\otimes N})$.
Before we start presenting conventional transformation schemes, we need to make a few remarks on transformed Hamiltonians and notations pertaining to them. First of all, we identify the set of gates $\{\mathcal{P}, \mathbb{I}\}^{\otimes n} = \{X, Y, Z, \mathbb{I}\}^{\otimes n}$ with the term Pauli strings (on n qubits). The previously mentioned Jordan-Wigner transform, obviously has the power to transform (2.10) into a Hamiltonian that is a weighted sum of Pauli strings on N qubits. General transforms, however, might involve other types of gates. We however have the choice to decompose these into Pauli strings. One might want to do so when using standard techniques for Hamiltonian simulation. In the following, we will denote the correspondence of second quantized operators or states B to their qubit counterparts C by: $B \mathrel{\hat{=}} C$. For convenience, we will also omit identities in Pauli strings and

rather introduce qubit labels, e.g. $X \otimes \mathbb{I} \otimes X = X_1 \otimes X_3 = (\bigotimes_{i\in\{1,3\}} X_i)$ and write $\mathbb{I}^{\otimes n} = \mathbb{I}$. A complete table of notations can be found in Section 2.8.

Consider a linear encoding of N fermionic sites into $n = N$ qubits given by a binary matrix A such that

$$|\boldsymbol{\omega}\rangle = |\mathbf{e}(\boldsymbol{\nu})\rangle = |A\boldsymbol{\nu}\rangle \hat{=} \left(\prod_{j=1}^{N} (c_j^\dagger)^{\nu_j}\right) |\Theta\rangle \tag{2.11}$$

and A is invertible, i.e. $\left(AA^{-1} \bmod 2\right) = \mathbb{I}$. Note that in this case, the decoding given by $\boldsymbol{\nu} = \mathbf{d}(\boldsymbol{\omega}) = \left(A^{-1}\boldsymbol{\omega}\right)$ is also linear. It is known that any such matrix A, subsequently also yields a mapping of the fermionic creation and annihilation operators to qubit gates [27]. To see how these are constructed, let us start by noting that they must fulfill the properties given in (2.2)-(2.5) and (1.7), which motivates the definition of a parity, a flip and an update set below:

1. $c_{i_m}^{(\dagger)}$ anticommutes with the first $m-1$ operators and thus acquires the phase $(-1)^{m-1}$.

2. A creation operator $c_{i_m}^\dagger$ might be absent (present) in between $c_{i_{m-1}}^\dagger$ and $c_{i_{m+1}}^\dagger$, leading the rightmost operator $c_{i_m}^{(\dagger)}$ to map the entire state to zero since $c_{i_m} |\Theta\rangle = 0 \left(c_{i_m}^\dagger c_{i_m}^\dagger = 0\right)$.

3. Given that the state was not annihilated, the occupation of site i_m has to be changed. This means a creation operator $c_{i_m}^\dagger$ has to be added or removed between $c_{i_{m-1}}^\dagger$ and $c_{i_{m+1}}^\dagger$.

These rules tell us what the transform of an operator $c_j^{(\dagger)}$ has to inflict on a basis state (2.11). In order to implement the phase shift of the first rule, a series of Pauli-Z operators is applied on qubits, whose numbers are in the *parity set* (with respect to $j \in [N]$), $P(j) \subseteq [N]$. Following the second rule we project onto the ± 1 subspace of the Z-string on qubits indexed by another $[N]$ subset, the so-called *flip set* of j, $F(j)$. The *update set* of j, $U(j) \subseteq [N]$ labels the qubits to be flipped completing the third rule using

an X-string.

$$(c_j^\dagger)^b (c_j)^{b+1 \bmod 2} \;\hat{=}\; \frac{1}{2}\left(\bigotimes_{k\in U(j)} X_k\right)\left(\mathbb{I} - (-1)^b \bigotimes_{l\in F(j)} Z_l\right)\bigotimes_{m\in P(j)} Z_m\,, \tag{2.12}$$

with $b \in \mathbb{Z}_2$. $P(j)$, $F(j)$ and $U(j)$ depend on the matrices A and A^{-1} as well as the parity matrix R. The latter is a $(N \times N)$ binary matrix which has its lower triangle filled with ones, but not its diagonal. For the matrix entries this means $R_{ij} = \theta_{ij}$, with θ_{ij} as the discrete version of the Heaviside function

$$\theta_{ij} = \begin{cases} 0 & i \leq j \\ 1 & i > j\,, \end{cases} \qquad R = \begin{bmatrix} 0 & & & & \\ 1 & 0 & & & \\ 1 & 1 & 0 & & \\ 1 & 1 & 1 & 0 & \\ \vdots & \vdots & \vdots & \ddots & \ddots \end{bmatrix}. \tag{2.13}$$

The set members are obtained in the following fashion:

1. $P(j)$ contains all column numbers in which the j-th row of matrix RA^{-1} has non-zero entries.

2. $F(j)$ contains the column labels of non-zero entries in the j-th row of A^{-1}.

3. $U(j)$ contains all row numbers in which the j-th column of A has non-zero entries.

Note that this definition of the sets differs from their original appearance in [27, 37], where diagonal elements are not included. In this way, our sets are not disjoint, which leads to Z-cancellations and appearance of Pauli-Y operators, but we have generalized the sets for arbitrary invertible matrices, and provided a pattern for other transforms later.

2.3.1 Jordan-Wigner, Parity and Bravyi-Kitaev transform

As an illustration, we present popular examples of these linear transformations, note again that all of these will have $n = N$. The Jordan-Wigner

transform is a special case for $A = \mathbb{I}$, leading to the direct mapping. The operator transform gives $L = O(N)$ Pauli strings as

$$(c_j^\dagger)^b (c_j)^{b+1 \bmod 2} \mathrel{\hat{=}} \frac{1}{2}\left(X_j + i(-1)^b Y_j\right) \bigotimes_{m<j} Z_m \,. \tag{2.14}$$

In the parity transform [27], we have $L = O(N)$ X-strings:

$$A^{-1} = \begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ & \ddots & \ddots & \\ & & 1 & 1 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ \vdots & \vdots & \ddots & \\ 1 & 1 & \cdots & 1 \end{bmatrix}, \tag{2.15}$$

$$(c_j^\dagger)^b (c_j)^{b+1 \bmod 2} \;\hat{=}\; \frac{1}{2}\left(Z_{j-1} \otimes X_j - i(-1)^b Y_j\right) \bigotimes_{m=j+1}^{N} X_m \,. \tag{2.16}$$

The Bravyi-Kitaev transform [26] is defined by a matrix A [27, 37] that has non-zero entries according to a certain binary tree rule, achieving $L = O(\log N)$.

2.4 Encoding only a subspace

2.4.1 Saving qubits by exploiting symmetries

Our goal is to be able to trade quantum resources, which is done by reducing degrees of freedom by exploiting symmetries. For that purpose, we provide a theoretical foundation to characterize the latter.
Parity, Jordan-Wigner and Bravyi-Kitaev transforms encode all $\mathcal{F}_N$ states and provide mappings for every $\mathcal{L}(\mathcal{F}_N)$ operator. Unfortunately, they require us to own a N-qubit quantum computer, which might be unnecessary. In fact, the only operator we want to simulate is the Hamiltonian, which usually has certain symmetries. Taking these symmetries into account enables us to perform the same task with $n \leq N$ qubits instead. Symmetries usually divide the $\mathcal{F}_N$ into subspaces, and the idea is to encode only one of those. Let $\mathcal{B}$ be a basis spanning a subspace $\mathrm{span}(\mathcal{B}) \subseteq \mathcal{F}_N$ be associated with a Hamiltonian (2.10), where for every l, $\boldsymbol{a}$, $\boldsymbol{b}$; $\widehat{h}_{\boldsymbol{ab}} : \mathrm{span}(\mathcal{B}) \mapsto \mathrm{span}(\mathcal{B})$. Usually, Hamiltonian symmetries generate many such (distinct) subspaces. Under consideration of

additional information about our problem, like particle number, parity or spin polarization, we select the correct subspace. Note that particle number conservation is by far the most prominent symmetry to take into account. It is generated by Hamiltonians that are linear combinations of products of $c_i^\dagger c_j \,|\, i,j \in [N]$. These Hamiltonians, originating from first principles, only exhibit terms conserving the total particle number; $\widehat{h}_{\boldsymbol{ab}} : \mathcal{H}_N^M \mapsto \mathcal{H}_N^M$. From all the Hilbert spaces $\mathcal{H}_N^M$, one considers the space with the particle number matching the problem description.
These symmetries will be utilized in the next section: we develop a language that allows for encodings $\boldsymbol{e}$ that reduce the length of the binary vectors $\boldsymbol{e}(\boldsymbol{\nu})$ as compared to $\boldsymbol{\nu}$. This means that the state $\boldsymbol{\nu}$ will be encoded in $n \leq N$ qubits, since each bit saved corresponds to a qubit eliminated. As suggested by Bravyi et al. [34], qubit savings can be achieved under the consideration of non-square, invertible matrices A. However, we will see below that using transformations based on nonlinear encodings and decodings $\boldsymbol{d}$ (the inverse transform defined by A^{-1} before), we can eliminate a number of qubits that scales with the system size. For linear codes on the other hand, we find a mere constant saving.

2.4.2 General transforms

We here show how second-quantized operators and states, Hamiltonian symmetries and the fermionic basis $\mathcal{B}$ are fused into a simple description of occupation basis states. While in this section all general ideas are presented, we would like to refer the reader to the appendices for details: to Section 2.7.1 in particular, which holds the proof of the underlying techniques. Fermionic basis states are represented by binary vectors $\boldsymbol{\nu} \in \mathbb{Z}_2^{\otimes N}$, with its components implicating the occupation of the corresponding orbitals. Basis states inside the quantum computer, on the other hand, are represented by binary vectors on a smaller space $\boldsymbol{\omega} \in \mathbb{Z}_2^{\otimes n}$. These vectors are code words of the former $\boldsymbol{\nu}$, where the binary code connecting all $\boldsymbol{\nu}$ and $\boldsymbol{\omega}$ is possibly nonlinear. In the end, an instance of such a code will be sufficient to describe states and operators, in a similar way than the matrix pair (A, A^{-1}) governs the conventional transforms already presented. We now start by defining such codes and connect them to the state mappings.
Let $\mathrm{span}\,(\mathcal{B})$ be a subspace of $\mathcal{F}_N$, as defined previously. For $n \geq \log|\mathcal{B}|$, we define two binary vector functions $\boldsymbol{d} : \mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2^{\otimes N}$, $\boldsymbol{e} : \mathbb{Z}_2^{\otimes N} \mapsto \mathbb{Z}_2^{\otimes n}$,

where we regard each component $\boldsymbol{d} = (d_1, \ldots, d_N)^\top$ as a binary function $d_i : \mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2$. Furthermore we introduce the binary basis set $\mathcal{V} \subseteq \mathbb{Z}_2^{\otimes N}$, with

$$\boldsymbol{\nu} \in \mathcal{V}, \quad \text{only if} \quad \left(\prod_{i=1}^{N} (c_i^\dagger)^{\nu_i} \right) |\Theta\rangle \in \mathcal{B} \,. \tag{2.17}$$

All elements in $\mathcal{B}$ shall be represented in $\mathcal{V}$. If for all $\boldsymbol{\nu} \in \mathcal{V}$ the binary functions $\boldsymbol{e}$ and $\boldsymbol{d}$ satisfy $\boldsymbol{d}(\boldsymbol{e}(\boldsymbol{\nu})) = \boldsymbol{\nu}$, and for all $\boldsymbol{\omega} \in \mathbb{Z}_2^{\otimes n} : \boldsymbol{d}(\boldsymbol{\omega}) \in \mathcal{V}$, then we call the two functions encoding and decoding, respectively. An encoding-decoding pair ($\boldsymbol{e}$, $\boldsymbol{d}$) forms a code.
We thus have obtained a general form of encoding, in which qubit states only represent the subspace $\mathrm{span}(\mathcal{B})$. The decoding, on the other hand, translates the qubit basis back to the fermionic one:

$$|\boldsymbol{\omega}\rangle = \bigotimes_{j=1}^{n} |\omega_j\rangle \;\hat{=}\; \left(\prod_{i=1}^{N} (c_i^\dagger)^{d_i(\boldsymbol{\omega})} \right) |\Theta\rangle \;. \tag{2.18}$$

We intentionally keep the description of these functions abstract, as the code used might be nonlinear, i.e. it cannot be described with matrices A, A^{-1}. Nonlinearity is thereby predominantly encountered in decoding rather than in encoding functions, as we will see in the examples obtained later.
For any code ($\boldsymbol{e}$, $\boldsymbol{d}$), we will now present the transform of fermionic operators into qubit gates. Before we can do so however, two issues are to be addressed. Firstly, one observes that we cannot hope to find a transformation recipe for a singular fermionic operator $c_j^{(\dagger)}$. The reason for this is that the latter operator changes the occupation of the j-th orbital. As a consequence, a state with the occupation vector $\boldsymbol{\nu}$ is mapped to $\boldsymbol{\nu} + \boldsymbol{u_j}$, where $\boldsymbol{u_j}$ is the unit vector of component j; $(u_j)_i = \delta_{ij}$. The problem is that since we have trimmed the basis, $\boldsymbol{\nu} + \boldsymbol{u_j}$ will probably not be in $\mathcal{V}$, which means this state is not encoded[1]. The action of $c_j^{(\dagger)}$ is, thus, not defined. We can however obtain a recipe for the non-vanishing Hamiltonian terms $\widehat{h}_{\boldsymbol{ab}}$ as they do not escape the encoded space being $(\mathrm{span}(\mathcal{B}) \mapsto \mathrm{span}(\mathcal{B}))$-operators. Note that this issue is never

[1] 'Unencoded state' is actually a slightly misleading term: when we say a state $\boldsymbol{\lambda} \in \mathbb{Z}_2^{\otimes N}$ is not encoded, we actually mean that it cannot be encoded and correctly decoded, so $\boldsymbol{d}(\boldsymbol{e}(\boldsymbol{\lambda})) \neq \boldsymbol{\lambda}$.

encountered in the conventional transforms, as they encode the entire Fock space.
Secondly, we are yet to introduce a tool to transform fermionic operators into quantum gates. The structure of the latter has to be similar to the linear case, as they mimic the same dynamics as presented in Section 2.3. In general, a gate sequence will commence with some kind of projectors into the subspace with the correct occupation, as well as operators implementing parity phase shifts. The sequence should close with bit flips to update the state. The task is now to determine the form of these operators. The issue boils down to finding operators that extract binary information from qubit states, and map it onto their phase. In other words, we need to find linear operators associated with e.g. the binary function d_j, such that it maps basis states $|\boldsymbol{\omega}\rangle \mapsto (-1)^{d_j(\boldsymbol{\omega})} |\boldsymbol{\omega}\rangle$. In any case, we must recover the case of Pauli strings on their respective sets when considering linear codes. For our example, this means the linear case yields the operator $(\bigotimes_{m\in F(j)} Z_m)$. Using general codes, we are lead to define the extraction superoperation $\mathfrak{X}$, which maps binary functions to quantum gates on n qubits:

$$\mathfrak{X} : \left(\mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2\right) \mapsto \mathcal{L}\left((\mathbb{C}^2)^{\otimes n}\right) . \tag{2.19}$$

The extraction superoperator is defined for all binary vectors $\boldsymbol{\omega} \in \mathbb{Z}_2^{\otimes n}$ and binary functions $f, g : \mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2$ as:

$$\mathfrak{X}[f]\,|\boldsymbol{\omega}\rangle = (-1)^{f(\boldsymbol{\omega})}\,|\boldsymbol{\omega}\rangle$$
(Extraction property) (2.20)

$$\mathfrak{X}\left[\boldsymbol{\omega} \mapsto f(\boldsymbol{\omega}) + g(\boldsymbol{\omega})\right] = \mathfrak{X}[f]\;\mathfrak{X}[g]$$
(Exponentiation identity) (2.21)

$$\mathfrak{X}\left[\boldsymbol{\omega} \mapsto b\right] = (-1)^b\,\mathbb{I} \quad | \; b \in \mathbb{Z}_2$$
(Extracting constant functions) (2.22)

$$\mathfrak{X}\left[\boldsymbol{\omega} \mapsto \omega_j\right] = Z_j \quad | \; j \in [n]$$
(Extracting linear functions) (2.23)

$$\mathfrak{X}\left[\boldsymbol{\omega} \mapsto \prod_{j\in\mathcal{S}} \omega_j\right] = \mathrm{C}^k\,\textsc{Phase}(i_1,\ldots,i_{k+1})$$
$$\text{with}\;\; \mathcal{S} = \{i_s\}_{s=1}^{k+1} \subseteq [n],\;\; k \in [n-1]$$
$$\text{(Extracting nonlinear functions).} \tag{2.24}$$

Note that the first two properties imply that the operators $\mathfrak{X}[f]$, $\mathfrak{X}[g]$ commute and all operators are diagonal in the computational basis. Given that binary functions have a polynomial form, we are now able to construct operators by extracting every binary function possible, for example

$$\begin{aligned}
&\mathfrak{X}[\boldsymbol{\omega} \mapsto 1 + \omega_1 + \omega_1\omega_2] \\
&= \mathfrak{X}\left[\boldsymbol{\omega} \mapsto 1\right]\, \mathfrak{X}\left[\boldsymbol{\omega} \mapsto \omega_1\right]\, \mathfrak{X}\left[\boldsymbol{\omega} \mapsto \omega_1\omega_2\right] &(2.25)\\
&= -Z_1\,\textsc{CPhase}(1,2)\,. &(2.26)
\end{aligned}$$

We firstly we have used (2.21) to arrive at (2.25), and then reach (2.26) by applying the properties (2.22)-(2.24) to the respective sub-terms. This might however not be the final Hamiltonian, since the simulation algorithm might require us to reformulate the Hamiltonian as a sum of weighted Pauli strings [38, 39]. In that case, need to decompose all controlled gates. The cost for this decomposition is an increase in the number of Hamiltonian terms, for instance we find $\textsc{CPhase}(i,j) = \frac{1}{2}(\mathbb{I} + Z_i + Z_j - Z_i \otimes Z_j)$. In general, (2.23) and (2.24) can be replaced by an adjusted definition:

$$\mathfrak{X}\left[\boldsymbol{\omega} \mapsto \prod_{j\in\mathcal{S}} \omega_j\right] = \mathbb{I} - 2\prod_{j\in\mathcal{S}} \frac{1}{2}\left(\mathbb{I} - Z_j\right) \quad \Big|\; \mathcal{S} \subseteq [n]$$
$$\text{(Extracting non-constant functions).} \tag{2.27}$$

We will be able to define the operator mappings introducing the parity and update functions, $\boldsymbol{p}$ and $\boldsymbol{\varepsilon}^{\boldsymbol{q}}$:

$$\boldsymbol{p} : \mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2^{\otimes N}, \quad p_j\left(\boldsymbol{\omega}\right) = \sum_{i=1}^{j-1} d_i\left(\boldsymbol{\omega}\right)\,, \tag{2.28}$$
$$\boldsymbol{\varepsilon}^{\boldsymbol{q}} : \mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2^{\otimes n}, \quad \text{with } \boldsymbol{q} \in \mathbb{Z}_2^{\otimes N}$$
$$\boldsymbol{\varepsilon}^{\boldsymbol{q}}\left(\boldsymbol{\omega}\right) = \boldsymbol{e}\left(\boldsymbol{d}\left(\boldsymbol{\omega}\right) + \boldsymbol{q}\right) + \boldsymbol{\omega}\,. \tag{2.29}$$

Finally, we have collected all the means to obtain the operator mapping for weight-l operator sequences as they occur in (2.10):

$$\prod_{i=1}^{l}(c_{a_i}^\dagger)^{b_i}(c_{a_i})^{1+b_i \bmod 2} \;\hat{=}\; \mathcal{U}^{\boldsymbol{a}}\left(\prod_{v=1}^{l-1}\prod_{w=v+1}^{l}(-1)^{\theta_{a_v a_w}}\right) \times \prod_{x=1}^{l}\frac{1}{2}\left(\mathbb{I}-\left[\prod_{y=x+1}^{l}(-1)^{\delta_{a_x a_y}}\right](-1)^{b_x}\,\mathfrak{X}\left[d_{a_x}\right]\right)\mathfrak{X}\left[p_{a_x}\right] \tag{2.30}$$

where θ_{ij} is defined in (2.13) and δ_{ij} is the Kronecker delta. In this expression, we find various projectors, parity operators with corrections for occupations that have changed before the update operator is applied. The update operator $\mathcal{U}^{\boldsymbol{a}}$, is characterized by the $\mathbb{Z}_2^{\otimes N}$-vector $\boldsymbol{q} = \sum_{i=1}^{l}\boldsymbol{u}_{\boldsymbol{a}_i}$.

$$\mathcal{U}^{\boldsymbol{a}} = \sum_{\boldsymbol{t}\in\mathbb{Z}_2^{\otimes n}}\left[\bigotimes_{i=1}^{n}(X_i)^{t_i}\right]\prod_{j=1}^{n}\frac{1}{2}\left(\mathbb{I}+(-1)^{t_j}\,\mathfrak{X}\left[\varepsilon_j^{\boldsymbol{q}}\right]\right). \tag{2.31}$$

This is a problem: when summing over the entire $\mathbb{Z}_2^{\otimes n}$, one has to expect an exponential number of terms. As a remedy, one can arrange the resulting operations into controlled gates, or rely on codes with a linear encoding. If the encoding can be defined using a binary $(n \times N)$-matrix A, $\boldsymbol{e}(\boldsymbol{\nu}) = A\boldsymbol{\nu}$, the update operator reduces to

$$\mathcal{U}^{\boldsymbol{a}} = \bigotimes_{i=1}^{n}(X_i)^{\sum_j A_{ij}q_j}\ . \tag{2.32}$$

In Section 2.7.1, we show that (2.30)-(2.32) satisfy the conditions (1.7)-(2.5). Note that the update operator is also important for state preparation: let us assume that our qubits are initialized all in their zero state, $(\bigotimes_{i\in[n]}|0\rangle)$, then the fermionic basis state associated with the vector $\boldsymbol{\nu}$ is obtained by applying the update operator $\mathcal{U}^{\boldsymbol{a}}$. Here the vector $\boldsymbol{a}$ contains all occupied orbitals, such that $\boldsymbol{q} = \boldsymbol{\nu}$. Even for nonlinear encodings the state preparation can done with Pauli strings: as the initial state is a product state of all zeros, we can replace operators $\mathfrak{X}[\boldsymbol{\omega} \mapsto \prod_{i\in\mathcal{S}\subseteq[n]}\omega_i]$ by $\mathbb{I}$.

In the following we will turn our attention to the most fruitful symmetry to take into account: particle conservation symmetry. While code families accounting for this symmetry are explored in the next subsection, alternatives to the mapping of entire Hamiltonian terms are discussed for such codes in Section 2.7.2.

2.4.3 Particle number conserving codes

In the following, we will present three types of codes that save qubits by exploiting particle number conservation symmetry, and possibly the conservation of the total spin polarization. Particle number conserving Hamiltonians are highly relevant for quantum chemistry and problems posed from first principles. We therefore set out to find codes in which $\boldsymbol{\nu} \in \mathcal{V}$ have a constant Hamming weight $\mathrm{w_H}(\boldsymbol{\nu}) = K$. Since the Hamming weight is defined as $\mathrm{w_H}(\boldsymbol{\nu}) = \sum_m \nu_m$, where the sum is defined without the modulus, it yields the total occupation number for the vectors $\boldsymbol{\nu}$. In order to simulate systems with a fixed particle number, we are thus interested to find codes that implement code words of constant Hamming weight. Note that the fixed Hamming weight K does not necessarily need to coincide with the total particle number M. A code with the such a property might also be interesting for systems with additional symmetries. Most importantly, we have not taken into account the spin multiplicity yet. As the particles in our system are fermions, every spatial site will typically have an even number of spin configurations associated with it. Orbitals with the same spin configurations naturally denote subsets of the total amount of orbitals, much like the suits in a card deck. An absence of magnetic terms as well as spin-orbit interactions leaves the Hamiltonian to conserve the number of particles inside all those suits. Consequently, we can append several constant-weight codes to each other. Each of those subcodes encodes thereby the orbitals inside one suit. In electronic system with only Coulomb interactions for instance, we can use two subcodes ($\boldsymbol{e}^{\diamondsuit}$, $\boldsymbol{d}^{\diamondsuit}$) and ($\boldsymbol{e}^{\spadesuit}$, $\boldsymbol{d}^{\spadesuit}$), to encode all spin-up, and spin-down orbitals, respectively. The global code ($\boldsymbol{e}$, $\boldsymbol{d}$), encoding the entire system, is obtained by appending the subcode functions e.g. $\boldsymbol{d}\left(\boldsymbol{\omega^1} \oplus \boldsymbol{\omega^2}\right) = \boldsymbol{d}^{\diamondsuit}(\boldsymbol{\omega^1}) \oplus \boldsymbol{d}^{\spadesuit}(\boldsymbol{\omega^2})$. Appending codes like this will help us to achieve higher savings at a lower gate cost.
The codes that we now introduce (see also again Table 2.1), fulfill the task of encoding only constant-weight words differently well. The larger $\mathcal{V}$, the less qubits will be eliminated, but we expect the resulting gate sequences to be more simple. Although not just words of that weight are encoded, we treat K as a parameter - the targeted weight.

2.4.3.1 Checksum codes

A slim, constant amount of qubits can be saved with the following $n = N - 1$, affine linear codes. Checksum codes encode all the words with either even or odd Hamming weight. As this corresponds to exactly half of the Fock space, one qubit is eliminated. This means we disregard the last component when we encode $\boldsymbol{\nu}$ into words with one digit less. The decoding function then adds the missing component depending on the parity of the code words. The code for K odd is defined as

$$\boldsymbol{d}(\boldsymbol{\omega}) = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ 1 & \cdots & 1 \end{bmatrix} \boldsymbol{\omega} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}, \tag{2.33}$$

$$\boldsymbol{e}(\boldsymbol{\nu}) = \begin{bmatrix} 1 & & & 0 \\ & \ddots & & \vdots \\ & & 1 & 0 \end{bmatrix} \boldsymbol{\nu}. \tag{2.34}$$

In the even-K version, the affine vector $\boldsymbol{u_N}$, added in the decoding, is removed. Since encoding and decoding function are both at most affine linear, the extracted operators will all be Pauli strings, with at most a minus sign. The advantage of the checksum codes is that they do not depend on K. They can be used even in cases of smaller saving opportunities, like $K \approx N/2$. We can employ these codes even for Hamiltonians that conserve only the fermion parity. This makes them important for effective descriptions of superconductors [40].

2.4.3.2 Codes with binary addressing

We present a concept for heavily nonlinear codes for large qubit savings, $n = \lceil \log(N^K/K!) \rceil$, [31]. In order to conserve the maximum amount of qubits possible, we choose to encode particle coordinates as binary numbers in $\boldsymbol{\omega}$. To keep it simple, we here consider the example of weight-one binary addressing codes, and refer the reader to Section 2.7.3 for $K > 1$. In $K = 1$, we recognize the qubit savings to be exponential, so consider $N = 2^n$. Encoding and decoding functions are defined by means of the

binary enumerator, $\mathrm{bin} : \mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}$, with $\mathrm{bin}\,(\boldsymbol{\omega}) = \sum_{j=1}^{n} 2^{j-1}\omega_j$.

$$d_j\,(\boldsymbol{\omega}) = \prod_{i=1}^{n} \left(\omega_i + 1 + q_i^j\right)\,, \tag{2.35}$$

$$\boldsymbol{e}\,(\boldsymbol{\nu}) = \left[\begin{array}{c|c|c|c} \boldsymbol{q^1} & \boldsymbol{q^2} & \cdots & \boldsymbol{q^{2^n}} \end{array}\right] \boldsymbol{\nu}\,, \tag{2.36}$$

where $\boldsymbol{q^j} \in \mathbb{Z}_2^{\otimes n}$ is implicitly defined by $\mathrm{bin}(\boldsymbol{q^j}) + 1 = j$. An input $\boldsymbol{\omega}$ will by construction render only the j-th component of (2.35) non-zero, when $\boldsymbol{q^j} = \boldsymbol{\omega}$.
The exponential qubit savings come at a high cost: the product over each component of $\boldsymbol{\omega}$ implies multi-controlled gates on the entire register. This is likely to cause connectivity problems. Note that decomposing the controlled gates will in general be practically prohibited by the sheer amount of resulting terms. On top of those drawbacks, we also expect the encoding function to be nonlinear for $K > 1$.

2.4.3.3 Segment codes

We introduce a type of scaleable $n = \lceil N/(1 + \frac{1}{2K})\rceil$ codes to eliminate a linear amount of qubits. The idea of segment codes is to cut the vectors $\boldsymbol{\nu}$ into smaller, constant-size vectors $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}} \in \mathbb{Z}_2^{\otimes \widehat{N}}$, such that $\boldsymbol{\nu} = \bigoplus_i \widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$. Each such segment $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$ is encoded by a subcode. Although we have introduced the concept already, this segmentation is independent from our treatment of spin 'suits'. In order to construct a weight-K global code, we append several instances of the same subcode. Each of these subcodes codes is defined on $\widehat{n}$ qubits, encoding $\widehat{N} = \widehat{n} + 1$ orbitals. We deliberately have chosen to only save one qubit per segment in order to keep the segment size $\widehat{N}(K)$ small.
We now turn our attention to the construction of these segment codes. As shown in Section 2.7.4, the segment sizes can be set to $\widehat{n} = 2K$ and $\widehat{N} = 2K + 1$. As the global code is supposed to encode all $\boldsymbol{\nu} \in \mathbb{Z}_2^{\otimes N}$ with Hamming weight K, each segment must encode all vectors from Hamming weight zero up to weight K. In this way, we guarantee that the encoded space contains the relevant, weight-K subspace. This construction follows from the idea that each block contains equal or less than K

particles, but might as well be empty. For each segment, the following de- and encoding functions are found for $\widehat{\boldsymbol{\omega}} \in \mathbb{Z}_2^{\otimes \widehat{n}}$, $\widehat{\boldsymbol{\nu}} \in \mathbb{Z}_2^{\otimes \widehat{N}}$:

$$\widehat{\boldsymbol{d}}\left(\widehat{\boldsymbol{\omega}}\right) = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ 0 & \dots & 0 \end{bmatrix} \widehat{\boldsymbol{\omega}} \; + \; f\left(\widehat{\boldsymbol{\omega}}\right) \begin{pmatrix} 1 \\ \vdots \\ \vdots \\ 1 \end{pmatrix} \tag{2.37}$$

$$\widehat{\boldsymbol{e}}\left(\widehat{\boldsymbol{\nu}}\right) = \begin{bmatrix} 1 & & & 1 \\ & \ddots & & \vdots \\ & & 1 & 1 \end{bmatrix} \widehat{\boldsymbol{\nu}}\,, \tag{2.38}$$

where $f : \mathbb{Z}_2^{\otimes \widehat{n}} \mapsto \mathbb{Z}_2$ is a binary switch. The switch is the source of nonlinearity in these codes. On an input $\widehat{\boldsymbol{\omega}}$ with $\mathrm{w_H}\left(\widehat{\boldsymbol{\omega}}\right) > K$, it yields one, and zero otherwise.
There is just one problem: segment codes are not suitable for particle-number conserving Hamiltonians, according to the definition of the basis $\mathcal{B}$, that we would have for segment codes. The reason for this is that we have not encoded all states with $\mathrm{w_H}\left(\boldsymbol{\nu}\right) > K$. In this way, Hamiltonian terms $\widehat{h}_{\boldsymbol{ab}}$ that exchange occupation numbers between two segments, can map into unencoded space. We can, however, adjust these terms, such that they only act non-destructively on states with at most K particles between the involved segment. This does not change the model, but aligns the Hamiltonian with the necessary condition that we have on $\mathcal{B}$, $\widehat{h}_{\boldsymbol{ab}} : \mathrm{span}(\mathcal{B}) \mapsto \mathrm{span}(\mathcal{B})$. This is discussed in detail Section 2.7.4, where we also provide an explicit description of the binary switch mentioned earlier.

Using segment codes, the operator transforms will have multi-controlled gates as well: the binary switch is nonlinear. However, gates are controlled on at most an entire segment, which means there is no gate that acts on more than $2K$ qubits. This an improvement in gate locality, as compared to binary addressing codes.

2.5 Examples

2.5.1 Hydrogen molecule

In this subsection, we will demonstrate the Hamiltonian transformation on a simple problem. Choosing a standard example, we draw comparison with other methods for qubit reduction. As one of the simplest problems, the minimal electronic structure of the hydrogen molecule has been studied extensively for quantum simulation [23, 38] already. We describe the system as two electrons on 2 spatial sites. Because of the spin-multiplicity, we require 4 qubits to simulate the Hamiltonian in conventional ways. Using the particle conservation symmetry of the Hamiltonian, this number can be reduced. The Hamiltonian also lacks terms that mix spin-up and -down states, with the total spin polarization known to be zero in the ground state. Taking into account these symmetries, one finds a total of 4 fermionic basis states:
$\mathcal{V} = \{(0,1,0,1),(0,1,1,0),(1,0,0,1),(1,0,1,0)\}$. These can be encoded into two qubits by appending two instances of a ($N = 2$, $n = 1$, $K = 1$)-code. The global code is defined as :

$$\boldsymbol{d}(\boldsymbol{\omega}) = \begin{bmatrix} 1 & \\ 1 & \\ & 1 \\ & 1 \end{bmatrix} \boldsymbol{\omega} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \tag{2.39}$$

$$\boldsymbol{e}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{\nu}\,. \tag{2.40}$$

The physical Hamiltonian,

$$\begin{aligned} H = & -h_{11}\left(c_1^\dagger c_1 + c_3^\dagger c_3\right) - h_{22}\left(c_2^\dagger c_2 + c_4^\dagger c_4\right) \\ & + h_{1331}\, c_1^\dagger c_3^\dagger c_3 c_1 + h_{2442}\, c_2^\dagger c_4^\dagger c_4 c_2 \\ & + h_{1221}\left(c_1^\dagger c_4^\dagger c_4 c_1 + c_3^\dagger c_2^\dagger c_2 c_3\right) \\ & + (h_{1221} - h_{1212})\left(c_1^\dagger c_2^\dagger c_2 c_1 + c_3^\dagger c_4^\dagger c_4 c_3\right) \\ & + h_{1212}\left(c_1^\dagger c_4^\dagger c_3 c_2 + c_2^\dagger c_3^\dagger c_4 c_1\right) \\ & + h_{1212}\left(c_1^\dagger c_3^\dagger c_4 c_2 + c_2^\dagger c_4^\dagger c_3 c_1\right), \end{aligned} \tag{2.41}$$

is transformed into the qubit Hamiltonian

$$g_1 \, \mathbb{I} + g_2 \, X_1 \otimes X_2 + g_3 \, Z_1 + g_4 \, Z_2 + g_5 \, Z_1 \otimes Z_2 \, . \tag{2.42}$$

The real coefficients g_i are formed by the coefficients h_{ijkl} of (2.41). After performing the transformation, we find

$$g_1 = -h_{11} - h_{22} + \frac{1}{2}h_{1221} + \frac{1}{4}h_{1331} + \frac{1}{4}h_{2442} \tag{2.43}$$

$$g_2 = h_{1212} \tag{2.44}$$

$$g_3 = g_4 = \frac{1}{2}h_{11} - \frac{1}{2}h_{22} + -\frac{1}{4}h_{1331} + \frac{1}{4}h_{2442} \tag{2.45}$$

$$g_5 = -\frac{1}{2}h_{1221} + \frac{1}{4}h_{1331} + \frac{1}{4}h_{2442} \, . \tag{2.46}$$

In previous works, conventional transforms have been applied to that problem Hamiltonian. Afterwards, the resulting 4-qubit-Hamiltonian has been reduced by hand in some way. In [41], the actions on two qubits are replaced with their expectation values after inspection of the Hamiltonian. In [33], on the other hand, the Hamiltonian is reduced to two qubits in a systematic fashion. Finally, the case is revisited in [34], where the problem is reduced below the combinatorial limit to one qubit. The latter two attempts have used Jordan-Wigner, the former the Bravyi-Kitaev transform first.

2.5.2 Fermi-Hubbard model

We present another example to illustrate the trade-off between qubit number and gate cost as well as circuit depth. For that purpose, we consider a simple toy Hamiltonian and demonstrate that a reduction of qubit requirements is theoretically possible. Although we do not want to claim that this scenario is realistic, we present a simple cost model with it, that hints the potential up-scaling of circuit depth and simulation cost, as the number of qubits decreases: we therefore consider the total sum of Pauli lengths of every term, which gives us an idea of the number of two-qubit gates required, and the number of Hamiltonian terms, as we decompose controlled gates (2.27), which should give us an idea of possible T-gate requirements and simulation depth. Let us start now to describe the model. We consider a small lattice with periodic boundary conditions in the lateral direction. The system shall contain 10 spatial sites, doubled by the

spin-multiplicity. The problem Hamiltonian is

$$H = -t \sum_{\langle i,j\rangle \in E} \left(c_i^\dagger c_j + c_j^\dagger c_i \right) + U \sum_{j=1}^{10} c_j^\dagger c_j \, c_{10+j}^\dagger c_{10+j} \,, \tag{2.47}$$

with its real coefficients t, U. It exhibits hopping terms along the edges E of the graph in Figure 2.1. The sketch on the left of this figure shows the connection graph of the first 10 orbitals. The other 10 orbitals are connected in the same fashion, and each such site is interacting with its counterpart from the other graph. We aim to populate this model with four fermions, where the total spin polarization is zero. Two conventional transforms and two transforms based on our codes are compared by the amount of qubits necessary, as well as the size of the transformed Hamiltonian. Note that besides eigenenergies, one might also be interested in obtaining the values of correlation functions, e.g. $\langle c_i^\dagger c_j \rangle$, which is done by measuring (qubit) operators obtained with the transform (2.47). The only difference is that if a correlator maps into unencoded space, it is to be set to zero. As benchmarks, we decompose controlled gates and count the number of resulting Pauli strings. The sum of their total weight constitutes the gate count. Having these two disconnected graphs is an invitation to us to append two codes acting on sites $1-10$ and $11-20$ respectively. For this example, we consider the following codes:

1. Jordan-Wigner and Bravyi-Kitaev transform: for comparison, we employ these conventional transforms on our system, with which we do not save qubits. The resulting terms are best obtained by the transforming every fermion operator in (2.47) by (2.12), where the flip, parity and update sets, $F(j)$, $P(j)$, $U(j)$ are determined by the choice of matrices A and A^{-1}, which are binary-tree matrices in the case of the Bravyi-Kitev transform, and identity matrices for the Jordan-Wigner transform.

2. Checksum code $\oplus$ checksum code: knowing that the particle number is conserved, and that spin cannot be flipped, we are free to save 2 qubits in constraining the parity of both, spin-up and -down particles, alike. This is done in appending two (N=10) checksum codes, where each that acts on only spin-up (spin-down) orbitals,

so indices 1 to 10 (11 to 20). The code resulting from appending two even checksum codes is linear, and encoding and decoding function feature the matrices A, A^{-1} as

$$\left[\begin{array}{ccc|ccc} 1 & & 0 & & & \\ & \ddots & \vdots & & & \\ & 1 & 0 & & & \\ \hline & & & 1 & & 0 \\ & & & & \ddots & \vdots \\ & & & & 1 & 0 \end{array}\right], \quad \left[\begin{array}{ccc|ccc} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ 1 & \cdots & 1 & & & \\ \hline & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \\ & & & 1 & \cdots & 1 \end{array}\right]. \tag{2.48}$$

However, as not the entire Fock-space is encoded, we need to perform the operator transform according to (2.30), where the update operator is defined by (2.32), where A refers here to the first matrix.

3. Segment code $\oplus$ segment code: Knowing the particle number in one 'spin suite' to be 2, we can for both, spin-up and -down orbitals, append two $K = 2$ segment codes to each other. This equals a total of 4 segment codes, saving 4 qubits. The resulting global code $(\boldsymbol{e}, \boldsymbol{d})$ is defined by

$$\boldsymbol{e}\left(\bigoplus_{i=1}^{4} \widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}\right) = \bigoplus_{i=1}^{4} \widehat{\boldsymbol{e}}(\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}), \tag{2.49}$$

$$\boldsymbol{d}(\boldsymbol{\omega}) = \left(\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & & & 1 \\ & \ddots & & \vdots \\ & & 1 & 1 \end{bmatrix}\right) \boldsymbol{\omega}, \tag{2.50}$$

where $\widehat{\boldsymbol{e}}$ are the encodings of the subcodes (2.38), and $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$ are occupations on the segments of the total orbital vector $\boldsymbol{\nu} = \bigoplus_i \widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$. These segments are formed as suggested by the right-hand side of Figure 2.1. For details on the decoding functions and Hamiltonian adjustments, please consider Section 2.7.4. The Hamiltonian transform is in the end carried out again by (2.30) and (2.32).

4. Checksum code $\oplus$ segment code: a compromise between the above, in which the spin-up orbitals are transformed via a checksum code, and the spin-down orbitals are transformed via two segment codes. The global code used for the Hamiltonian transformation is the

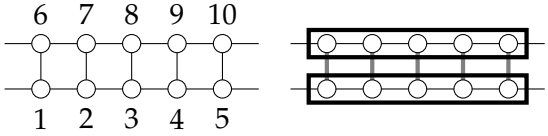

Figure 2.1. Left: illustration of the Fermi-Hubbard model considered. Lines between two sites, like 1 and 2, indicate the appearance of the term $t(c_1^\dagger c_2 + c_2^\dagger c_1)$ in the Hamiltonian (2.47). Periodic boundary conditions link sites 1 and 5 as well as 6 and 10. Sites 11-20 follow the same graph. Right: segmenting of the system; the two blocks are infringed. The gray links are to be adjusted.

appendage of an (even-weight, $N = 10$) checksum code and two $(K = 2)$ segment codes, including Hamiltonian adjustments on the spin-down orbitals.

Note that from the combinatorial perspective, we could encode the problem with 11 qubits. However, if we append two $K = 2$ binary addressing codes to each other, the resulting Hamiltonian is on 14 qubits already. The problem is that the resulting Hamiltonian for this case cannot be expressed with decomposed controlled gates due to the high number of resulting terms.

Indeed, Table 2.2 suggests that decomposing the controlling gates might easily lead to very large Hamiltonians with a multitude of very small terms. The gate decomposition appears therefore undesirable. We in general recommend to rather decompose large controlled gates as shown in [42]. However, one also notices that an elimination of up to two qubits comes at a low cost: the amount of gates is not higher than in the Bravyi-Kitaev transform. As soon as we employ segment codes on the other hand, the Hamiltonian complexity rises with the amount of qubits eliminated.

2.6 Conclusion

In this chapter, we have introduced new methods to reduce the number of qubits required for simulating fermionic systems in second quantization. We see the virtue of the introduced concepts in the fact that it takes into account symmetries on a simple but non-abstract level. We merely

Mapping	Qubits	Weight	Terms
Jordan-Wigner transform	20	232	74
Bravyi-Kitaev transform	20	278	74
Checksum code $\oplus$ Checksum code	18	260	74
Checksum code $\oplus$ Segment code	17	4425	876
Segment code $\oplus$ Segment code	16	9366	1838

Table 2.2. Relaxing the qubit requirements for the Hamiltonian (2.47), where various mappings trade different amounts of qubits. The notation $\oplus$ is used as two codes for different graphs are appended. We compare different mappings by the amount of qubits. We make comparrisons by the number of Hamiltonian terms and the total weight of the resulting Pauli strings.

concern ourselves with objects as simple as binary vectors, but attribute the physical interpretation of orbital occupations to them. At this level, the mentioned symmetries are easy to apply and exploit. The accounting for the complicated antisymmetrization of the many-body wave function on the other hand is done in the fermionic operators, which to transform we have provided recipes for. In these operator transforms we see room for improvement: we for instance lack a proper gate composition for update operators of nonlinear encodings at this point. We on the other hand have the extraction superoperator $\mathfrak{X}$ return only conventional (multi)-controlled phase gates. Nonlinear codes would on the other hand benefit from a gate set that includes gates with negative control, i.e. with the (-1) eigenvalue conditioned on $|0\rangle$ eigenspaces of certain qubits involved. We consider our work to be relevant for quantum simulation with near-term devices, with a limited number of qubits at disposal. Remarks about asymptotic scaling are thus missing in this work, but would be interesting. Also, we have centered our investigations around quantum computers with qubits. The idea behind the generalized operator transforms, however, can possibly be adapted to multi-level systems (qudits). The operator transforms of segment and binary addressing codes, for instance, might simplify in such a setup, if generalized Pauli operators are available in some form.

Apart from the codes presented, we have laid the foundation for the reader to invent their own. For that purpose, we have added the functionality of defining and using binary code transforms (with linear encoding functions $\boldsymbol{e}$) to the *OpenFermion* software package [11].

2.7 Supplement

2.7.1 General operator mappings

The goal of this section is to verify that the fermionic mode is accurately represented by our qubit system. This is divided into three steps: step one is to analyze the action of Hamiltonian terms on the fermionic basis. In the second step, we verify parity and projector parts of (2.30) to work like the original operators in step one, disregarding the occupational update for a moment. Conditions for this state update are subsequently derived. The update operator (2.31) is shown to fulfill these conditions in the third step, thus concluding the proof.

2.7.1.1 Hamiltonian dynamics

In order to verify that the gate sequences (2.30) are mimicking the Hamiltonian dynamics adequately, we verify that the resulting terms have the same effect on the Hamiltonian basis. This is done on the level of second quantization with respect to the notation (2.17): no transition into a qubit system is made. This step serves the sole purpose to quantify the effect of the Hamiltonian terms on the states. To that end, we begin by studying the effect of a singular fermionic operator $c_j^{(\dagger)}$ on a pure state, before considering an entire term $\widehat{h}_{\boldsymbol{ab}}$ on a state in $\mathcal{B}$. As a preliminary, we note that (2.2)-(2.5) follow directly from (1.7), when considering that

$$c_j c_j = c_j^\dagger c_j^\dagger = c_j \left|\Theta\right\rangle = 0\,. \tag{2.51}$$

The relations (2.2)-(2.5) indicate how singular operators act on pure states in general. We now become more specific and apply these rules to a state $(\prod_i (c_i^\dagger)^{\nu_i}) \left|\Theta\right\rangle$, that is not necessarily in $\mathcal{B}$, but is described by an occupation vector $\boldsymbol{\nu} \in \mathbb{Z}_2^{\otimes N}$. The effect of an annihilation operator on

such a state is considered first:

$$c_j \left[\prod_{i=1}^{N} \left(c_i^\dagger\right)^{\nu_i}\right] |\Theta\rangle \;=\; \left[\prod_{i<j} \left(-c_i^\dagger\right)^{\nu_i}\right] \; c_j \left(c_j^\dagger\right)^{\nu_j} \; \left[\prod_{k>j} \left(c_k^\dagger\right)^{\nu_k}\right] |\Theta\rangle \tag{2.52}$$

$$= \left[\prod_{i<j} \left(-c_i^\dagger\right)^{\nu_i}\right] \; \frac{1}{2}\left[1-(-1)^{\nu_j}\right] \; \left[\prod_{k>j} \left(c_k^\dagger\right)^{\nu_k}\right] |\Theta\rangle \tag{2.53}$$

$$= \left[\prod_{i<j} (-1)^{\nu_i}\right] \; \frac{1}{2}\left[1-(-1)^{\nu_j}\right] \; \left[\prod_{k=1}^{N} \left(c_k^\dagger\right)^{\nu_k+\delta_{jk} \bmod 2}\right] |\Theta\rangle \tag{2.54}$$

A short explanation on what has happened: in (2.52), c_j has anticommuted with all creation operator $c_i^\dagger$ that have indexes $i < j$. Depending on the component ν_j, a creation operator $c_j^\dagger$ might now be to the right of the annihilator c_j. If the creation operator is not encountered, we may continue the anticommutations of c_j until it meets the vacuum and annihilates the state by $c_j |\Theta\rangle = 0$. Using the anticommutation relations (1.7), we therefore replace $c_j (c_j^\dagger)^{\nu_j}$ with $\frac{1}{2}\left[1-(-1)^{\nu_j}\right]$ when going from (2.52) to (2.53). Finally, the terms are rearranged in (2.54): conditional sign changes of the anticommutations are factored out of the new state with an occupation that is now described by the binary vector $(\boldsymbol{\nu} + \boldsymbol{u_j})$ rather than $\boldsymbol{\nu}$. When considering to apply a creation operator $c_j^\dagger$ on the former state, the result is similar. Alone at step (2.53), we have to replace $c_j^\dagger (c_j^\dagger)^{\nu_j}$ by $\frac{1}{2}\left[1+(-1)^{\nu_j}\right]$ instead, as now the case of appearance of the creation operator leads to annihilation: $c_j^\dagger c_j^\dagger = 0$. We thus find

$$c_j^\dagger \left[\prod_{i=1}^{N} \left(c_i^\dagger\right)^{\nu_j}\right] |\Theta\rangle = \left[\prod_{i<j} (-1)^{\nu_i}\right] \; \frac{1}{2}\left[1+(-1)^{\nu_j}\right] \; \left[\prod_{k=1}^{N} \left(c_k^\dagger\right)^{\nu_k+\delta_{jk} \bmod 2}\right] |\Theta\rangle \,. \tag{2.55}$$

We now turn our attention to the actual goal, the effect that a Hamiltonian term from (2.10) has on a state in $\mathcal{B}$ (this means its occupation vector $\boldsymbol{\nu}$ is in $\mathcal{V}$). We therefore consider a generic operator sequence $\prod_{i=1}^{l} (c_{a_i}^\dagger)^{b_i} (c_{a_i})^{1+b_i \bmod 2}$, parametrized by some N-ary vector $\boldsymbol{a} \in [N]^{\otimes l}$ and a binary vector $\boldsymbol{b} \in \mathbb{Z}_2^{\otimes l}$, for some length l. With (2.54) and (2.55), we now have the means to consider the effect such a sequence of annihilation and creation operators. The two relations will be repeatedly utilized in an inductive procedure, as every single operator $(c_{a_i}^\dagger)^{b_i} (c_{a_i})^{1+b_i \bmod 2}$

of $\prod_{i=1}^{l}(c_{a_i}^\dagger)^{b_i}(c_{a_i})^{1+b_i \bmod 2}$ will act on a basis state, one after another. The state's occupation is updated after every such operation. For convenience, we define:

$$\boldsymbol{\nu}^{(i)} \in \mathbb{Z}_2^{\otimes N} \quad \Big| \quad i \in \{0, \ldots, l\} \tag{2.56}$$

$$\boldsymbol{\nu}^{(l)} = \boldsymbol{\nu} \in \mathcal{V} \tag{2.57}$$

$$\boldsymbol{\nu}^{(i-1)} = \boldsymbol{\nu}^{(i)} + \boldsymbol{u_{a_i}} \ . \tag{2.58}$$

Now, the procedure starts:

$$\left[\prod_{i=1}^{l}(c_{a_i}^\dagger)^{b_i}(c_{a_i})^{1+b_i \bmod 2}\right] \left[\prod_{k=1}^{N}(c_k^\dagger)^{\nu_k}\right]|\Theta\rangle \tag{2.59}$$

$$= \left[\prod_{i=1}^{l-1}(c_{a_i}^\dagger)^{b_i}(c_{a_i})^{1+b_i \bmod 2}\right] \frac{1}{2}\left[1-(-1)^{b_l}(-1)^{\nu_{a_l}}\right]$$
$$\times (-1)^{\sum_{j<a_l}\nu_j}\left[\prod_{k=1}^{N}\left(c_k^\dagger\right)^{\nu_k+\delta_{a_l k} \bmod 2}\right]|\Theta\rangle \tag{2.60}$$

$$= \left[\frac{1}{2}\left[1-(-1)^{b_l}(-1)^{\nu_{a_l}^{(l)}}\right](-1)^{\sum_{j<a_l}\nu_j^{(l)}}\right]$$
$$\times \left[\prod_{i=1}^{l-1}\left(c_{a_i}^\dagger\right)^{b_i}(c_{a_i})^{1+b_i \bmod 2}\right] \left[\prod_{k=1}^{N}(c_k^\dagger)^{\nu_k^{(l-1)}}\right]|\Theta\rangle \tag{2.61}$$

$$= \left[\prod_{i=1}^{l} \underbrace{\frac{1}{2}\left[1-(-1)^{b_i}(-1)^{\nu_{a_i}^{(i)}}\right]}_{\text{projector eigenvalues}} \underbrace{(-1)^{\sum_{j<a_i}\nu_j^{(i)}}}_{\text{parity signs}}\right] \underbrace{\left[\prod_{k=1}^{N}(c_k^\dagger)^{\nu_k^{(0)}}\right]|\Theta\rangle}_{\text{updated state}} \tag{2.62}$$

We again explain what has happened: first, the rightmost operator, which is either c_{a_l} or $c_{a_l}^\dagger$ depending on the parameter b_l, acts on the state according to either (2.54) or (2.55). We therefore combine the two relations for the absorption of this operator $(c_{a_l}^\dagger)^{b_l}(c_{a_l})^{1+b_l \bmod 2}$ in (2.60). In the same fashion, all the remaining operators of the sequence are one-after-another absorbed into the state. The new state is described by the vector $\boldsymbol{\nu}^{(l-1)}$ after the update. And the cycle begins anew with $(c_{a_{l-1}}^\dagger)^{b_{l-1}}(c_{a_{l-1}})^{1+b_{l-1} \bmod 2}$. From (2.61) on, we use the notations (2.56)-(2.58) to describe partially updated occupations. By the end of this iteration, the occupation of the state is changed to $\boldsymbol{\nu}^{(0)} = \boldsymbol{\nu} + \boldsymbol{q}$, with the total change $\boldsymbol{q} = \sum_i \boldsymbol{u_{a_i}}$. Also, the coefficients of (2.62) take into account sign changes from anticommutations ("parity signs" in (2.62)) and the eigenvalues of the applied projec-

tions. In its entirety, (2.62) denotes the resulting state, and is the main ingredient for the next step.

2.7.1.2 Parity operators and projectors

We are given the operator transform (2.30) and the state transform (2.18). We want to show the that the fermion system is adequately simulated, which means to show that the effect (2.62) is replicated by (2.30) acting on $|\boldsymbol{e}(\boldsymbol{\nu})\rangle$. This is the goal of the next two steps. We start by evaluating the application of (2.30) on that state, up to the update operator $\mathcal{U}^{\,a}$. This means that the operators applied implement two things only: the parity signs of (2.62), and the projection onto the correct occupational state. Note that these parity operators and projectors are applied before the update operator in (2.30):

$$\overbrace{\mathcal{U}^{\,\boldsymbol{a}}}^{\text{update operator}} \overbrace{\left(\prod_{v=1}^{l-1}\prod_{w=v+1}^{l}(-1)^{\theta_{a_v a_w}}\right)}^{\text{parity signs}} \times \prod_{x=1}^{l} \underbrace{\frac{1}{2}\left(\mathbb{I}-\left[\prod_{y=x+1}^{l}(-1)^{\delta_{a_x a_y}}\right](-1)^{b_x}\,\mathfrak{X}\,[d_{a_x}]\right)}_{\text{projectors}} \underbrace{\mathfrak{X}\,[p_{a_x}]}_{\text{parity operators}} . \tag{2.63}$$

We now commence our evaluation:

$$\mathcal{U}^{\,\boldsymbol{a}}\left[\left(\prod_{v=1}^{l-1}\prod_{w=v+1}^{l}(-1)^{\theta_{a_v a_w}}\right) \prod_{x=1}^{l}\frac{1}{2}\left(\mathbb{I}-\left[\prod_{y=x+1}^{l}(-1)^{\delta_{a_x a_y}}\right](-1)^{b_x}\,\mathfrak{X}\,[d_{a_x}]\right)\mathfrak{X}\,[p_{a_x}]\right]|\boldsymbol{e}\,(\boldsymbol{\nu})\rangle \tag{2.64}$$

$$= \mathcal{U}^{\,\boldsymbol{a}}\left[\left(\prod_{v=1}^{l-1}\prod_{w=v+1}^{l}(-1)^{\theta_{a_v a_w}}\right) \prod_{x=1}^{l}\frac{1}{2}\left(1-\left[\prod_{y=x+1}^{l}(-1)^{\delta_{a_x a_y}}\right](-1)^{b_x}\,(-1)^{d_{a_x}(\boldsymbol{e}(\boldsymbol{\nu}))}\right)(-1)^{p_{a_x}(\boldsymbol{e}(\boldsymbol{\nu}))}\right]|\boldsymbol{e}\,(\boldsymbol{\nu})\rangle \tag{2.65}$$

$$= \mathcal{U}^{\boldsymbol{a}} \left[\left(\prod_{v=1}^{l-1} \prod_{w=v+1}^{l} (-1)^{\theta_{a_v a_w}} \right) \right.$$
$$\left. \prod_{x=1}^{l} \frac{1}{2} \left(1 - \left[\prod_{y=x+1}^{l} (-1)^{\delta_{a_x a_y}} \right] (-1)^{b_x} \, (-1)^{\nu_{a_x}} \right) (-1)^{\sum_{j<a_x} \nu_j} \right] |\boldsymbol{e}\,(\boldsymbol{\nu})\rangle \quad (2.66)$$

$$= \mathcal{U}^{\boldsymbol{a}} \left[\prod_{x=1}^{l} \frac{1}{2} \left(1 - (-1)^{b_x} \, (-1)^{\nu_{a_x} + \sum_{y=x+1}^{l} \delta_{a_x a_y}} \right) \right.$$
$$\left. (-1)^{\sum_{j<a_x} \nu_j + \sum_{y=x+1}^{l} \theta_{a_x a_y}} \right] \quad |\boldsymbol{e}\,(\boldsymbol{\nu})\rangle \quad (2.67)$$

$$= \left[\prod_{x=1}^{l} \frac{1}{2} \left(1 - (-1)^{b_x} \, (-1)^{\nu_{a_x}^{(x)}} \right) (-1)^{\sum_{j<a_x} \nu_j^{(x)}} \right] \quad \mathcal{U}^{\boldsymbol{a}} \; |\boldsymbol{e}\,(\boldsymbol{\nu})\rangle \; . \quad (2.68)$$

Let us describe what has happened: in (2.65), the extraction property (2.20) is used, and we arrive at (2.66) after using the property $\boldsymbol{d}\,(\boldsymbol{e}\,(\boldsymbol{\nu})) = \boldsymbol{\nu}$ and the definition of the parity function. From there we go to (2.67) when we merge the two products and perform rearrangements that make it easy to cast all delta and theta functions into the components of the partially updated occupations $\boldsymbol{\nu}^{(i)}$, (2.68).
Comparing (2.68) to (2.62), we notice to have successfully mimicked the same sign changes and and projections, as the coefficients in both relations match. Now it is only left to show that the state update is executed correctly. Naively, one would think that we would need to show that

$$\mathcal{U}^{\boldsymbol{a}} \; |\boldsymbol{e}\,(\boldsymbol{\nu})\rangle \hat{=} \left[\prod_{k=1}^{N} \left(c_k^\dagger \right)^{\nu_k^{(0)}} \right] |\Theta\rangle \; , \quad (2.69)$$

but this is too strong a statement. It is in fact sufficient to demand

$$\mathcal{U}^{\boldsymbol{a}} |\boldsymbol{e}\,(\boldsymbol{\nu})\rangle \; = \; \left| \boldsymbol{e}\left(\boldsymbol{\nu}^{(0)} \right) \right\rangle \; = \; |\boldsymbol{e}\,(\boldsymbol{\nu} + \boldsymbol{q})\rangle \; . \quad (2.70)$$

For $\boldsymbol{\nu}^{(0)} \in \mathcal{V}$, (2.69) and (2.70) is equivalent. However, it might be the case that $\boldsymbol{\nu}^{(0)} \notin \mathcal{V}$, so $\boldsymbol{\nu}^{(0)}$ is not encoded. This mean that (2.69) is not fulfilled, since $\boldsymbol{d}(\boldsymbol{e}(\boldsymbol{\nu}^{(0)})) \neq \boldsymbol{\nu}^{(0)}$. It is however not necessary to include $\boldsymbol{\nu}^{(0)}$ in the encoding, as for $\boldsymbol{\nu}^{(0)} \notin \mathcal{V}$, the state will vanish anyways: we know from $\widehat{h}_{\boldsymbol{ab}} : \mathrm{span}(\mathcal{B}) \mapsto \mathrm{span}(\mathcal{B})$, that in this case $\widehat{h}_{\boldsymbol{ab}}$ must act destructively on

that basis state, $\widehat{h}_{\boldsymbol{ab}} \left(\prod_k (c_k^\dagger)^{\nu_k}\right) |\Theta\rangle = 0$. This detail is implemented by the projector part of the transformed sequence (2.30). These projectors are, as we have just shown, working faithfully like (2.62), for the transformed sequence acting on every $|\boldsymbol{\nu}\rangle$ with $\boldsymbol{\nu} \in \mathcal{V}$. Hence (2.70) is a sufficient condition for the updated state. The proof is completed once we have verified that (2.70) is satisfied with the update operator defined as in (2.31). This is done during the next step.

2.7.1.3 Update operator

The missing piece of the proof is to check that (2.31) and (2.32) fulfill the condition (2.70). We start by verifying the condition (2.70) for (2.32), which we have presented as special case of (2.31) for linear encoding functions: $\boldsymbol{e}\left(\boldsymbol{\nu} + \boldsymbol{\nu'}\right) = \boldsymbol{e}\left(\boldsymbol{\nu}\right) + \boldsymbol{e}\left(\boldsymbol{\nu'}\right)$. Using that property, one can in fact derive (2.32) from (2.31) directly. We now apply (2.32) to $|\boldsymbol{e}(\boldsymbol{\nu})\rangle$, but firstly we note that

$$X_j \left|\boldsymbol{\omega}\right\rangle = \left|\boldsymbol{\omega} + \boldsymbol{u_j}\right\rangle \ , \tag{2.71}$$

where $\boldsymbol{u_j}$ is the j-th unit vector of $\mathbb{Z}_2^{\otimes n}$. Using (2.71) and the linearity of $\boldsymbol{e}$, we find:

$$\begin{aligned}
\mathcal{U}^{\boldsymbol{a}} \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right\rangle &= \left[\bigotimes_{i=1}^{n} (X_i)^{\sum_j A_{ij} q_j}\right] |\boldsymbol{e}(\boldsymbol{\nu})\rangle && (2.72)\\
&= \left[\bigotimes_{i=1}^{n} (X_i)^{\boldsymbol{e}(\boldsymbol{q})}\right] |\boldsymbol{e}(\boldsymbol{\nu})\rangle && (2.73)\\
&= |\boldsymbol{e}(\boldsymbol{\nu}) + \boldsymbol{e}(\boldsymbol{q})\rangle && (2.74)\\
&= |\boldsymbol{e}(\boldsymbol{\nu} + \boldsymbol{q})\rangle \ , && (2.75)
\end{aligned}$$

which shows (2.70) for linear encodings.
We now turn our attention to general encodings and prove the same ex-

pression for update operators as defined in (2.31):

$$
\begin{aligned}
&\mathcal{U}^{\boldsymbol{a}} \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right\rangle \\
&= \left(\sum_{\boldsymbol{t} \in \mathbb{Z}_2^{\otimes n}} \left[\bigotimes_{i=1}^{n} \left(X_i\right)^{t_i} \right] \prod_{j=1}^{n} \frac{1}{2} \left(\mathbb{I} + (-1)^{t_j} \, \mathfrak{X}\left[\varepsilon_j^{\boldsymbol{q}}\right] \right) \right) \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right\rangle && (2.76) \\
&= \left(\sum_{\boldsymbol{t} \in \mathbb{Z}_2^{\otimes n}} \left[\bigotimes_{i=1}^{n} \left(X_i\right)^{t_i} \right] \prod_{j=1}^{n} \underbrace{\frac{1}{2} \left(1 + (-1)^{t_j + \varepsilon_j^{\boldsymbol{q}}(\boldsymbol{e}(\boldsymbol{\nu}))} \right)}_{\delta_{t_j \varepsilon_j^{\boldsymbol{q}}(\boldsymbol{e}(\boldsymbol{\nu}))}} \right) \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right\rangle && (2.77) \\
&= \left(\bigotimes_{i=1}^{n} \left(X_i\right)^{\varepsilon_i^{\boldsymbol{q}}(\boldsymbol{e}(\boldsymbol{\nu}))} \right) \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right\rangle && (2.78) \\
&= \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right) + \boldsymbol{\varepsilon}^{\boldsymbol{q}}\left(\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right)\right\rangle && (2.79) \\
&= \left|\boldsymbol{e}\left(\boldsymbol{\nu}\right) + \boldsymbol{e}\left(\boldsymbol{\nu}\right) + \boldsymbol{e}\left(\boldsymbol{d}\left(\boldsymbol{e}\left(\boldsymbol{\nu}\right)\right) + \boldsymbol{q}\right)\right\rangle && (2.80) \\
&= \left|\boldsymbol{e}\left(\boldsymbol{\nu} + \boldsymbol{q}\right)\right\rangle \, , && (2.81)
\end{aligned}
$$

which completes the proof. We swiftly recap what has happened: in (2.76), we have plugged the definition of(2.31) into the left-hand side of (2.70). In between this equation and (2.77), we have evaluated the expectation values of the extracted operators $\mathfrak{X}[\varepsilon_j^{\boldsymbol{q}}]$. From that line to the next, the $\mathbb{Z}_2^{\otimes n}$-sum is collapsed over the condition $\boldsymbol{t} = \boldsymbol{\varepsilon}^{\boldsymbol{q}}(\boldsymbol{e}(\boldsymbol{\nu}))$. We go from (2.78) to (2.79) by applying (2.71). Once we insert the definition (2.29) into (2.79), it becomes obvious that the condition (2.70) is fulfilled. Thus, the entire operator transform is now proven.

2.7.2 Transforming particle-number conserving Hamiltonians

In this section, we examine the richest symmetry to exploit for qubit savings: particle conservation. We begin by introducing the most relevant class of Hamiltonians that exhibit this symmetry, but ultimately the main goal of this section is to simplify the operator transform for all such Hamiltonians. Motivated by the compartmentalized recipes of the conventional mappings, (2.12), we suggest alternatives to the transform (2.30), that do not depend on the sequence length l.

Let us start by noting how easy it is to state that a Hamiltonian the total number of particles: a Hamiltonian like (2.10), conserves the total number of particles when every term $\widehat{h}_{\boldsymbol{ab}}$ has as many creation operators as

it has annihilation operators. The lengths l, implicit in the sequences $\widehat{h}_{\boldsymbol{ab}}$ that occur in the Hamiltonian, are thereby determined by the field theory or model, that underlies the problem. The coefficients $h_{\boldsymbol{ab}}$, on the other hand, are determined by the set of basis functions used. For first-principle problems in quantum chemistry and solid state physics, we usually encounter particle-number-conserving Hamiltonians with terms of weight that is at most $l = 4$ (1.8). In the notation of (2.10), these coefficients V_{ijkl} and t_{ij} correspond to $h_{(i,j,k,l)(1,1,0,0)}$ and $h_{(i,j)(1,0)}$. The $(l = 4)$ interaction terms usually originate from either magnetism and/or the Coulomb interaction. Even for these $(l = 4)$-terms, the operator transform (2.30) is quite bulky, and we in general would like to have a transform that is independent of l. Before we begin to discuss such transform recipes however, we need to set up some preliminaries. First of all, we need to find a suitable code $(\boldsymbol{e}, \boldsymbol{d})$, as discussed in the main part. Ideally, we would encode only the Hilbert space with the correct number of particles, M, but Hilbert spaces of other particle numbers can also be included. Assuming that the Hamiltonian visits every state with the same particle number, we must encode entire Hilbert spaces $\mathcal{H}_N^m$ only. Secondly, we need to reorder the fermionic operators inside the Hamiltonian terms $\widehat{h}_{\boldsymbol{ab}}$. The reason for this is, that our goal can only be achieved by finding recipes for smaller sequences of constant length. In order to transform the Hamiltonian terms then, we need to invoke the anticommutation relations (1.7) to introduce an order in $\widehat{h}_{\boldsymbol{ab}}$, such that these small sequences appear as consecutive, distinct blocks. As we shall see, these blocks will have the shape $c_i^\dagger c_j$. So every $\widehat{h}_{\boldsymbol{ab}}$ needs to be reordered, such that every even operator is a creation operator, and every odd operator an annihilator. For the $(l = 4)$-terms in (1.8), this reordering means $c_i^\dagger c_j^\dagger c_k c_l \;\mapsto\; c_i^\dagger c_l\, c_j^\dagger c_k - \delta_{jl}\; c_i^\dagger c_k$.

Let us quickly sketch the idea behind that reordering and introduce some nomenclature: instead of considering Hamiltonian terms, we realize that also the terms $c_i^\dagger c_j$ also conserve the particle number: $\mathcal{H}_N^m \mapsto \mathcal{H}_N^m$. Let us act with $c_i^\dagger c_j$ on an encoded state. We consider a state that is not annihilated by $c_i^\dagger c_j$. Its particle number is reduced by one through c_j, but then immediately restored by $c_i^\dagger$. In fact, for a general sequence of that arrangement, every even operator restores the particle number in this way and every odd reduces it. We therefore call the subspace, in which we find the state after an even (odd) number of operators, the even (odd)

subspace. Since all l must be even for the Hamiltonian to have particle conservation symmetry, the even subspace is the one encoded. The odd subspace, on the other hand, has one particle less, so it is $\mathcal{H}_N^{(M-1)}$, if the even one is $\mathcal{H}_N^M$.

2.7.2.1 Encoding the two spaces separately

In this ordering, one can find a recipe for a singular creation or annihilation operator. The strategy is to consider a second code for the odd subspace. As before ($\boldsymbol{e}$, $\boldsymbol{d}$) denotes the code for the even subspace, and now ($\boldsymbol{e'}$, $\boldsymbol{d'}$) is encoding the odd subspace. The idea is that after an odd operator (which in this ordering is an annihilation operator), the state is updated into the odd subspace. With every even operator (which is a creation operator), the state is updated from the odd subspace back into the even one. We find:

$$c_j^\dagger \;\hat{=}\; \frac{1}{2}\,\bar{\mathcal{U}}^{(j)}\;(\mathbb{I}+\mathfrak{X}\,[d_j])\;\mathfrak{X}\,[p_j]\;, \tag{2.82}$$

$$c_j \;\hat{=}\; \frac{1}{2}\,\mathcal{U}^{(j)}\;\left(\mathbb{I}-\mathfrak{X}\left[d'_j\right]\right)\mathfrak{X}\left[p'_j\right]\;. \tag{2.83}$$

In (2.83), $\mathcal{U}^{(j)}$ is defined as in (2.31), but its counterpart from (2.82) is defined by

$$\bar{\mathcal{U}}^{(j)} = \sum_{\boldsymbol{t}\,\in\,\mathbb{Z}_2^{\otimes n}} \left[\bigotimes_{i=1}^{n}(X_i)^{t_i}\right] \prod_{i=1}^{n}\frac{1}{2}\left(\mathbb{I}+(-1)^{t_i}\;\mathfrak{X}\left[\varepsilon_k^{\prime\,\boldsymbol{u_j}}\right]\right)\;, \tag{2.84}$$

with the primed functions $\boldsymbol{\varepsilon'}^{\,\boldsymbol{q}}$, $\boldsymbol{p'}$ defined like (2.29) and (2.28), but with ($\boldsymbol{e'}$, $\boldsymbol{d'}$) in place of ($\boldsymbol{e}$, $\boldsymbol{d}$).
This method relies on n qubits being feasible to simulate the odd subspace in. That is, however, not always the case. The basis set of $\mathcal{H}_N^{M-1}$ is in general larger than $\mathcal{H}_N^M$, when $M > N/2$. In this way, the odd subspace can also be larger and even be infeasible to simulate with just n qubits. As a solution, one changes the ordering into odd operators being creation operators, and even ones being annihilators, like $c_k c_i^\dagger\, c_l\, c_j^\dagger$. This causes the odd subspace to become $\mathcal{H}_N^{(M+1)}$, which has a smaller basis set than $\mathcal{H}_N^M$. For that case ($\boldsymbol{e}$, $\boldsymbol{d}$) become the code for the odd subspace, and ($\boldsymbol{e'}$, $\boldsymbol{d'}$) will be associated to the even subspace in (2.82) and (2.83).
The obvious disadvantage is that two codes have to be employed at once.

However, the checksum code for instance (Section2.4.3.1 in the main part), comes in two different flavors already, which can be used as codes for even and odd subspaces, respectively.

2.7.2.2 Encoding the building blocks

The building blocks $c_i^\dagger c_j$ are guaranteed to conserve the particle number, so the even subspace is conserved. As a consequence, one may consider the possibility to transform the operators as the pairs we have rearranged them into. In this way, we still have a certain compartmentalization of (2.30). Two special cases are to be taken into account: when $i > j$, an additional minus sign has to be added, as compared to the $i < j$ case. Also, when $i = j$, all parity operators cancel and the projectors coincide. We find:

$$c_i^\dagger c_j \mathrel{\hat{=}} \begin{cases} \frac{1}{4}\,(-1)^{\theta_{ij}}\,\mathcal{U}^{(i,j)}\,\mathfrak{X}\left[p_i + p_j\right]\,\left(\mathbb{I} + \mathfrak{X}\left[d_i\right]\right)\left(\mathbb{I} - \mathfrak{X}\left[d_j\right]\right) & i \neq j \\ \\ \frac{1}{2}\left(1 - \mathfrak{X}\left[d_j\right]\right) & i = j\,, \end{cases} \tag{2.85}$$

with $\mathcal{U}^{(i,j)}$ being the $l = 2$ version of (2.31), and $\boldsymbol{p}$ and $\boldsymbol{\varepsilon}^{\boldsymbol{q}}$ defined as usual by (2.28) and (2.29).

2.7.3 Multi-weight binary addressing codes based on dissections

With binary addressing codes, that is codes that are similar to the one presented in Section 2.4.3.2 in the main part, even an exponential amount of qubits can be saved for systems with low particle number, but at the expense of complicated gates. For this section, we firstly recap the situation of Section 2.4.3.2 and clarify what binary addressing means. Firstly, some nomenclature is introduced. We then generalize the concept of binary addressing codes to weight-K codes, using results from [31]. As an example, we explicitly obtain the $K = 2$ code.

Suppose we have a system with $N = 2^r$ orbitals, and one particle in it. Our goal is to encode the basis state, where the particle is on orbital $y \in [2^r]$, as a binary number in r qubits. In this way, the state with occupational vector $\boldsymbol{u_y}$ is encoded as $|\boldsymbol{q}^{\boldsymbol{y,r}}\rangle$, with $\boldsymbol{q}^{\boldsymbol{y,r}} \in \mathbb{Z}_2^{\otimes r}$ and

$y = \mathrm{bin}(\boldsymbol{q}^{\boldsymbol{y},\boldsymbol{r}}) + 1$. Probing an unknown basis state, a decoding will now have components of the form

$$\boldsymbol{\omega} \mapsto \prod_{i\in[r]} (\omega_i + q_i^{y,r} + 1)\,. \tag{2.86}$$

Such binary functions output 1 only when $\boldsymbol{\omega} = \boldsymbol{q}^{\boldsymbol{y},\boldsymbol{r}}$. In our nomenclature, we say that in the basis state $|\boldsymbol{q}^{\boldsymbol{y},\boldsymbol{r}}\rangle$, the particle has the coordinate y. We refer to codes that store particle coordinates in binary form, as binary addressing codes.

In the $K = 1$ case from the main part, the code words just contain the binary representation of one coordinate. The question is now how to generalize the binary addressing codes. For multi-weight codes, we have to have K sub-registers to store the addresses of K particles. Naively, one would want to store the coordinate of each particle in its respective sub-register in binary form, as we have done for $K = 1$. This however, holds a problem. As the particles are indistinguishable, the stored coordinates would be interchangeable, the code would not be one-to-one. For the binary numbers $\boldsymbol{\omega}^1$ and $\boldsymbol{\omega}^2$, that represent a coordinate each, this would mean $\boldsymbol{d}(\boldsymbol{\omega}^1 \oplus \boldsymbol{\omega}^2) = \boldsymbol{d}(\boldsymbol{\omega}^2 \oplus \boldsymbol{\omega}^1)$. That strategy not only complicates the operator transform, it also leads to a certain qubit overhead, as each plain word has as many code words as there are permutations of K items. Since this naïve idea leaves us unconvinced, we abandon it and search for one-to-one codes instead. The key is to consider the coordinates to be in a certain format and this is where [31] comes into play. We proceed by using some relevant concepts of that paper.
Let us consider the coordinates of K particles to be given in the N-ary vector $\boldsymbol{x} = (x_1, \ldots, x_K)$. Between those coordinates, we have imposed an ordering $x_i > x_j$ as $i > j$. Particles cannot share the same orbital, so we are excluding the cases where two coordinates are equal. Using results from [31], we transform the latter into coordinates that lack such an ordering, and where each component is an integer from a different range:

$$\boldsymbol{x} \;\mapsto\; \boldsymbol{y} = (y_1, \ldots, y_K)^\top \qquad \text{with} \quad \boldsymbol{y} \in \bigotimes_{m=1}^{K} \left[\left\lceil \frac{N}{m} \right\rceil\right]\,. \tag{2.87}$$

Through that transform, each vector $\boldsymbol{y}$ corresponds to a valid vector $\boldsymbol{x}$,

and there is no duplication. We now represent the $\boldsymbol{y}$-coordinates by binary numbers in the code words $\boldsymbol{\omega} \in \mathbb{Z}_2^{\otimes n}$, where $n = \sum_{m=1}^{K} \lceil \log \frac{N}{m} \rceil$:

$$\boldsymbol{\omega} = \bigoplus_{m=1}^{K} \boldsymbol{q}^{\,y_m, \lceil \frac{N}{m} \rceil} \qquad \text{with} \quad \boldsymbol{q}^{\boldsymbol{i},\boldsymbol{j}} \in \mathbb{Z}_2^{\otimes j} \quad \text{and} \quad \mathrm{bin}\left(\boldsymbol{q}^{\boldsymbol{i},\boldsymbol{j}}\right) + 1 = i\,. \tag{2.88}$$

A geometric interpretation of the process portrays the vector $\boldsymbol{x}$ as a set of coordinates in a K-dimensional, discrete vector space. The vectors allowed by the ordering form thereby a multi-dimensional tetrahedron. The states outside the tetrahedron do not correspond to a valid $\mathcal{V}$ vector, so encoding each coordinate x_i in $\lceil \log N \rceil$ qubits would be redundant. We therefore dissect the tetrahedron, and rearrange it into a *brick*, as it is referred to in [31]. What is actually done is to apply symmetry operations (like point-reflections) on the vector space until the tetrahedron is deformed into the desired shape a K-dimensional, rectangular volume. The fact that the vectors to encode are now all inside a hyper-rectangle is what we wanted to achieve. We can now clip the ranges of the coordinate axes (to $[\lceil \log \frac{N}{m} \rceil]$) to exclude vectors the vectors outside the brick. As the values on the axes correspond to non-binary addresses, this means that the qubit space is trimmed as well, and we have eliminated all states based on not-allowed coordinates. This is where we now reconnect to our task of finding a code: the $\boldsymbol{e}$- and $\boldsymbol{d}$-functions have to take into account the reshaping process, as only the coordinates $\boldsymbol{x}$ have a physical interpretation and can be decoded. The binary addresses in the code words, on the other hand, are representatives of $\boldsymbol{y}$. With binary logic, the two coordinates have to be reconnected. We illustrate this abstract process on the example of the $(K = 2)$-code.

Weight-two binary addressing code

As an example, we present the weight-two binary addressing code on $N = 2^r$ orbitals. The integer r will determine the size of the entire qubit system $n = 2r - 1$, with two registers of size r and $r - 1$.

With the two registers, a binary vector $\boldsymbol{\omega} = \boldsymbol{\alpha} \oplus \boldsymbol{\beta}$ with $\boldsymbol{\alpha} \in \mathbb{Z}_2^{\otimes r}$ and $\boldsymbol{\beta} \in \mathbb{Z}_2^{\otimes (r-1)}$ is defining the qubit basis. In two dimensions, the brick turns into a rectangle and the tetrahedron into triangle. The decoding function takes binary addresses of the rectangular $\boldsymbol{y}$, and transforms them into coordinates in the triangle $\boldsymbol{x}$. The ordering condition implies hereby where to dissect the rectangle: Figure 2.2 may serve as a visual aid, disregard-

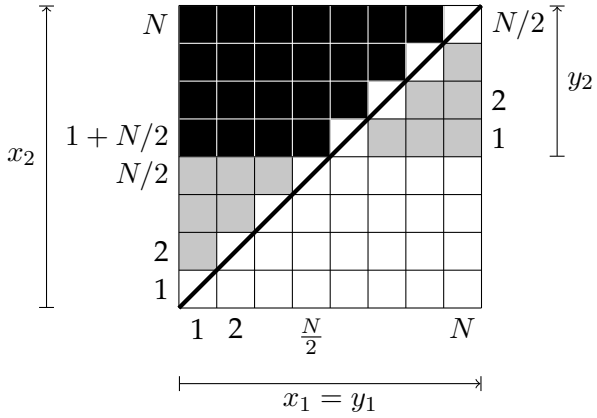

Figure 2.2. Visualization of the 2-dimensional vector space: a valid vector is represented as a colored tile. The left gray tiles and the black ones constitute the triangle, defining all valid vectors $\boldsymbol{x} = (x_1, x_2)^\top$. The marked diagonal tiles are to be excluded from the encoded space. The black tiles and the gray ones on the right of this diagonal form the brick, containing all $\boldsymbol{y} = (y_1, y_2)^\top$ vectors.

ing the excluded cases of $y_1 = y_2$, we find for $y_1 \in [N]$, $y_2 \in [N/2]$ and $\boldsymbol{x} \in [N]^{\otimes 2}$:

$$(x_1, x_2) = \begin{cases} (y_1, N/2 + y_2) & \text{for} \quad y_1 < N/2 + y_2 \\ (y_1, N/2 - y_2 + 1) & \text{for} \quad y_1 > N/2 + y_2 \,. \end{cases} \tag{2.89}$$

This decoding is translated into a binary functions as follows: the coordinate y_1 is represented by the binary vector $\boldsymbol{\alpha}$ and y_2 by $\boldsymbol{\beta}$. For each component defined by the binary vector $\boldsymbol{b} \in \mathbb{Z}_2^{\otimes r}$, we have

$$\begin{aligned} d_j(\boldsymbol{\alpha} \oplus \boldsymbol{\beta}) \;=\; & S(\boldsymbol{\alpha}, \boldsymbol{\beta}) \prod_{i=1}^{r} \left(\alpha_i + q_i^{j,r} + 1\right) \\ & + (1 + S(\boldsymbol{\alpha}, \boldsymbol{\beta}))(1 + T(\boldsymbol{\alpha}, \boldsymbol{\beta})) \prod_{i=1}^{r} \left(\alpha_i + q_i^{j,r}\right) \\ & + (1 + S(\boldsymbol{\alpha}, \boldsymbol{\beta}))(1 + T(\boldsymbol{\alpha}, \boldsymbol{\beta})) \prod_{k=1}^{r-1} \left(\beta_k + q_k^{j,r}\right) \\ & + S(\boldsymbol{\alpha}, \boldsymbol{\beta}) \prod_{k=1}^{r-1} \left(\beta_k + q_k^{j,r} + 1\right) , \end{aligned} \tag{2.90}$$

with $\boldsymbol{q}^{\boldsymbol{j,r}} = (q_1^{j,r}, q_2^{j,r}, \ldots, q_{2^r}^{j,r})$ as defined in (2.88) and we have employed two binary functions S and $T : (\mathbb{Z}_2^{\otimes r}, \mathbb{Z}_2^{\otimes(r-1)}) \mapsto \mathbb{Z}_2$. Here, S compares the binary numbers to determine if the coordinates are left of the dissection (a black tile in Figure 2.2).

$$S(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \alpha_r \sum_{j=1}^{r-1} \left[\prod_{r-1 \geq i > j} (\alpha_i + \beta_i + 1) \right] (1 + \alpha_j)\beta_j + 1 + \alpha_r \tag{2.91}$$

The binary function T, on the other hand, is checking whether a set of coordinates is on a diagonal position (diagonally marked tiles). These excluded cases are mapped to $(0)^{\otimes r}$ altogether.

$$T(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \prod_i (\alpha_i + \beta_i) \tag{2.92}$$

This concludes the decoding function. Unfortunately, the amount of logic elements in the decoding will complicate the weight-two codes quite a bit, and the encoding function is hardly better. The reason for this is to find in the ordering condition: the update operations are conditional on whether we change the ordering of the coordinates represented by $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. This is reflected in a nonlinear encoding function: we remind us that the encoding function is a map $\boldsymbol{e} : \mathbb{Z}_2^{\otimes 2^r} \mapsto \mathbb{Z}_2^{\otimes(2r-1)}$, and with $\boldsymbol{\nu} \in \mathbb{Z}_2^{\otimes 2^r}$ we find

$$\begin{aligned}
\boldsymbol{e}(\boldsymbol{\nu}) \quad = \quad & \sum_{j=2}^{2^{r-1}} \sum_{i=1}^{j-1} \left(\boldsymbol{q}^{\boldsymbol{i,r}} + \boldsymbol{I}^{\boldsymbol{r}} \right) \oplus \left(\boldsymbol{q}^{\boldsymbol{j,r-1}} + \boldsymbol{I}^{\boldsymbol{r-1}} \right) \nu_i \nu_j \\
& + \sum_{j=2^{r-1}+1}^{2^r} \sum_{i=1}^{2^{r-1}} \left(\boldsymbol{q}^{\boldsymbol{i,r}} + \boldsymbol{I}^{\boldsymbol{r}} \right) \oplus \left(\boldsymbol{q}^{\boldsymbol{j-2^{r-1},r-1}} \right) \nu_i \nu_j \\
& + \sum_{j=2^{r-1}+2}^{2^r} \sum_{i=2^{r-1}+1}^{j-1} \left(\boldsymbol{q}^{\boldsymbol{i,r}} \right) \oplus \left(\boldsymbol{q}^{\boldsymbol{j-2^{r-1},r-1}} \right) \nu_i \nu_j \,,
\end{aligned}$$

with $\boldsymbol{q}^{\boldsymbol{i,j}}$ as defined in (2.88), and $\boldsymbol{I}^{\boldsymbol{j}} = (1)^{\otimes j} = \boldsymbol{q}^{\boldsymbol{2^j,j}}$.
The dissecting of tetrahedrons can be generalized for codes of weight larger than two (see again [31]), but as one increases the number of dissections, the code functions are complicated even further.

2.7.4 Segment codes

In this section, we provide detailed information on the segment codes. We firstly concern ourselves with the segmentation of the global code, including a derivation of the segment sizes. In another subsection we construct the segment codes themselves. The last subsection is dedicated to the adjustments one has to make to Hamiltonian, such that segment codes become feasible to use.

2.7.4.1 Segment sizes

At this point we want to sketch the idea behind the segment sizes $(\widehat{N}, \widehat{n})$ stated during Section 2.4.3.3 in the main part, but first of all we would like to clearly set up the situation.

We consider vectors $\boldsymbol{\nu} \in \mathbb{Z}_2^{\otimes N}$ to consist of $\widehat{m}$ smaller vectors $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$ of length $\widehat{n}+1$, such that $\boldsymbol{\nu} = \bigoplus_{i=1}^{\widehat{m}} \widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$. We call those vectors $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$ segments of $\boldsymbol{\nu}$. The goal is now to find a code $(\boldsymbol{e}, \boldsymbol{d})$ to encode a basis $\mathcal{V}$ which contains all vectors $\boldsymbol{\nu}$ with Hamming weight K. For that purpose we relate the segment $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$ to a segment of the code space, $\widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}$, for all $i \in [N]$. The code space segments constitute the code words in a fashion similar to the previous segmentation of $\boldsymbol{\nu}$: $\boldsymbol{\omega} = \bigoplus_{i=1}^{\widehat{m}} \widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}$. However, the length of those binary vectors $\widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}$ is $\widehat{n}$, such that with $n = \widehat{m}\widehat{n}$ and $N = \widehat{m}(\widehat{n}+1)$, the problem is reduced by $\widehat{m}$ qubits as compared to conventional transforms. We now introduce the *subcodes* $(\widehat{\boldsymbol{e}} : \mathbb{Z}_2^{\otimes(\widehat{n}+1)} \mapsto \mathbb{Z}_2^{\otimes\widehat{n}}, \widehat{\boldsymbol{d}} : \mathbb{Z}_2^{\otimes\widehat{n}} \mapsto \mathbb{Z}_2^{\otimes(\widehat{n}+1)})$, with which we encode the i-th segment $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$ as $\widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}$ (see Figure 2.3). Note that we require the subcodes to inherit all the code properties. In this way we guarantee the code properties of the *global code* $(\boldsymbol{e}, \boldsymbol{d})$ when appending $\widehat{m}$ instances of the same subcode:

$$\boldsymbol{d}\left(\bigoplus_{i=1}^{\widehat{m}} \widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}\right) = \bigoplus_{i=1}^{\widehat{m}} \widehat{\boldsymbol{d}}\left(\widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}\right), \qquad \boldsymbol{e}\left(\bigoplus_{i=1}^{\widehat{m}} \widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}\right) = \bigoplus_{i=1}^{\widehat{m}} \widehat{\boldsymbol{e}}\left(\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}\right). \tag{2.93}$$

The orbital number being an integer multiple of the block size is of course an idealized scenario. One will probably have to add a few other components in order to compensate for dimensional mismatches.
We now set out to find the smallest segment size $\widehat{n}$. It should be clear that $\widehat{n}$ is a function of the targeted Hamming weight K: this means K determines which segment codes are suitable for the system. The reason for

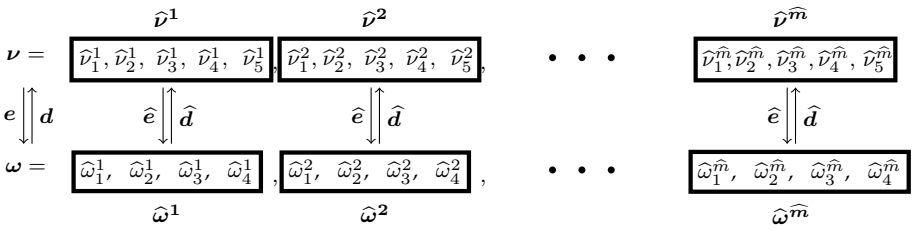

Figure 2.3. Visualization of (2.93) for $\widehat{n} = 4$. The global code ($\boldsymbol{e}$, $\boldsymbol{d}$) relates the occupation vectors to the global code words $\boldsymbol{\nu} \leftrightarrow \boldsymbol{\omega}$. The an instance of the subcode ($\widehat{\boldsymbol{e}}$, $\widehat{\boldsymbol{d}}$) relates i-th block in $\boldsymbol{\nu}$, $\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}$, to the i-th segment in the code words, $\widehat{\boldsymbol{\omega}}^{\boldsymbol{i}}$.

this is that we need to encode all vectors with weight 0 to K inside every segment, taking into account for the up to K particles on the orbitals inside one segment. In order to include weight-K vectors, the size of each segment must be at least K. If the segment size would be exactly K, on the other hand, we end up encoding the entire Fock space again. In doing so, we are not making any qubit savings. The segments must thus be larger than K. In other words, we look for an integer $\widehat{n} > K$, where the sum of all combinations $\widehat{\boldsymbol{\nu}} \in \mathbb{Z}_2^{\otimes(\widehat{n}+1)}$ with $\mathrm{w_H}\left(\widehat{\boldsymbol{\nu}}\right) \leq K$ is smaller equal $2^{\widehat{n}}$.

$$2^{\widehat{n}} \geq \sum_{k=0}^{K} \binom{\widehat{n}+1}{k} \tag{2.94}$$

In the case $\widehat{n} = 2K$, the condition is fulfilled as identity, since exactly half of all $2^{\widehat{n}+1}$ combinations are included in the sum.

2.7.4.2 Subcodes

This subsection offers a closer look at the construction of the segment subcodes ($\widehat{\boldsymbol{e}}$, $\widehat{\boldsymbol{d}}$). Let us start by considering the decoding $\widehat{\boldsymbol{d}}$ in order to explore the nature of the binary switch $f(\widehat{\boldsymbol{\omega}})$, that occurs in (2.37). One

observes the two (affine) linear $\left(\mathbb{Z}_2^{\otimes \widehat{n}} \mapsto \mathbb{Z}_2^{\otimes(\widehat{n}+1)}\right)$-maps

$$\widehat{\boldsymbol{\omega}} \;\mapsto\; \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ 0 & \dots & 0 \end{bmatrix} \widehat{\boldsymbol{\omega}}\,, \qquad \widehat{\boldsymbol{\omega}} \;\mapsto\; \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ 0 & \dots & 0 \end{bmatrix} \widehat{\boldsymbol{\omega}} \;+\; \begin{pmatrix} 1 \\ \vdots \\ \vdots \\ 1 \end{pmatrix}. \tag{2.95}$$

to produce together all the vectors with weight equal or smaller than K, if we input all $\widehat{\boldsymbol{\omega}}$ with $\mathrm{w_H}\left(\widehat{\boldsymbol{\omega}}\right) \leq K$ into the first, and the remaining cases with $\mathrm{w_H}\left(\widehat{\boldsymbol{\omega}}\right) > K$ into the second one. Note that the last component is always zero in outputs of the first function and one in the second. Therefore, the inverse of both maps is always a linear map with the matrix $\left[\,\mathbb{I} \,|\, \boldsymbol{I}^{\widehat{n}}\right]$. We take this inverse as encoding (2.38), and the two maps (2.95) are merged into the decoding (2.37). In order to switch between these two maps we define the binary function $f(\widehat{\boldsymbol{\omega}}) : \mathbb{Z}_2^{\otimes \widehat{n}} \mapsto \mathbb{Z}_2$ such that

$$f(\widehat{\boldsymbol{\omega}}) = \begin{cases} 1 & \text{for } \mathrm{w_H}\left(\widehat{\boldsymbol{\omega}}\right) > K \\ 0 & \text{otherwise}\,. \end{cases} \tag{2.96}$$

In general, one can define this binary switch in a brute-force way by

$$f\left(\widehat{\boldsymbol{\omega}}\right) = \sum_{k=K+1}^{2K} \sum_{\substack{\boldsymbol{t} \in \mathbb{Z}_2^{\otimes 2K} \\ \mathrm{w_H}(\boldsymbol{t})=k}} \prod_{m=1}^{2K} \left(\widehat{\omega}_m + 1 + t_m\right). \tag{2.97}$$

For the case $K = 1$ ($\widehat{n} = 2$), the switch equals $f(\boldsymbol{\omega}) = \omega_1\omega_2$, and for the code we recover a version of binary addressing codes, where the vector $(0, 0, 0)$ is encoded.

$$\widehat{\boldsymbol{d}}\left(\widehat{\boldsymbol{\omega}}\right) = \begin{pmatrix} \widehat{\omega}_1\left(\widehat{\omega}_2 + 1\right) \\ \left(\widehat{\omega}_1 + 1\right)\widehat{\omega}_2 \\ \widehat{\omega}_1\widehat{\omega}_2 \end{pmatrix}, \qquad \widehat{\boldsymbol{e}}\left(\widehat{\boldsymbol{\nu}}\right) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \widehat{\boldsymbol{\nu}}\,. \tag{2.98}$$

In the $K = 2$ ($\widehat{n} = 4$) case, this binary switch is found to be $f\left(\widehat{\boldsymbol{\omega}}\right) = \widehat{\omega}_1\widehat{\omega}_2\widehat{\omega}_3 + \widehat{\omega}_1\widehat{\omega}_2\widehat{\omega}_4 + \widehat{\omega}_1\widehat{\omega}_3\widehat{\omega}_4 + \widehat{\omega}_2\widehat{\omega}_3\widehat{\omega}_4 + \widehat{\omega}_1\widehat{\omega}_2\widehat{\omega}_3\widehat{\omega}_4$.

2.7.4.3 Hamiltonian adjustments

As mentioned in Section 2.4.3.3, in the main part, segment codes are not automatically compatible with all particle-number-conserving Hamiltonians. We show here, how certain adjustments can be made to these

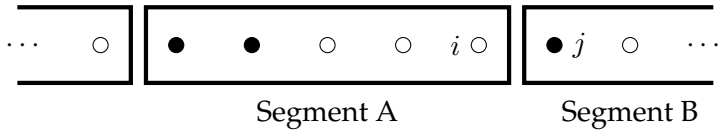

Figure 2.4. (Filled) Circles represent (occupied) fermionic orbitals, where $K = 2$ segment codes are used in the indicated blocks. This occupational case is problematic for the codes, as the operator $c_i^\dagger c_j$ acting on this state leaves the encoded space.

Hamiltonians, such that their action on the space $\mathcal{H}_N^K$ is not changed, but segment codes become feasible to describe them with. In order to understand this issue, we begin by examining the encoded space. For that purpose we reprise the situation of (2.93), where we have appended $\widehat{m}$ instances of the same subcode. With segment codes, the basis $\mathcal{V}$ contains vectors with Hamming weights from 0 to $\widehat{m}K$. We have encoded all possible vectors $\boldsymbol{\nu}$ with $0 \leq \mathrm{w_H}(\boldsymbol{\nu}) \leq K$, but although we have some, not all vectors with $\mathrm{w_H}(\boldsymbol{\nu}) > K$ are encoded. We can illustrate that point rather quickly: each segment has length $2K+1$, but the subcode encodes vectors $\widehat{\boldsymbol{\nu}}$ with only $\mathrm{w_H}(\widehat{\boldsymbol{\nu}}) \leq K$. The (global) basis $\mathcal{V}$ is thus deprived of vectors $\boldsymbol{\nu} = (\bigoplus_i \widehat{\boldsymbol{\nu}}^{\boldsymbol{i}})$ where for any segment i, $\mathrm{w_H}\left(\widehat{\boldsymbol{\nu}}^{\boldsymbol{i}}\right) > K$.
We now turn our attention to terms, which, when present in a Hamiltonian, make segment codes infeasible to use. Note, that $\mathcal{V}$-vectors with $\mathrm{w_H}(\boldsymbol{\nu}) \neq K$, are not corresponding to fermionic states we are interested in. In particular it is a certain subset of states with $\mathrm{w_H}(\boldsymbol{\nu}) > K$, which can lead out of the encoded space (into the states previously mentioned) when acted upon with certain fermionic operators. Let us consider the operator $c_i^\dagger c_j$ as an example, where i and j are in different segments (let us call these segments A and B). Now a basis state as depicted in Figure 2.4, is not annihilated by $c_i^\dagger c_j$, and leads into a state with 3 particles in segment A. The problem is that the initial state is encoded in the $(K = 2)$ segment codes, whereas the updated state (with the 3 particles in A) is not. In general, operators $\widehat{h}_{\boldsymbol{ab}}$, that change occupations in between segments, will cause some basis states with $\mathrm{w_H}(\boldsymbol{\nu}) > K$ to leave the encoded space. We can however adjust these terms $\widehat{h}_{\boldsymbol{ab}} \to \widehat{h}'_{\boldsymbol{ab}}$, such that $\widehat{h}'_{\boldsymbol{ab}} : \mathrm{span}(\mathcal{B}) \mapsto \mathrm{span}(\mathcal{B})$, where $\mathcal{B}$ is the basis encoded by the segment codes. We now sketch the idea behind those adjustments, before we reconsider the situation of Figure 2.4. Note that after these adjustments

have been made to all Hamiltonian terms in question, the segment codes are compatible with the new Hamiltonian. The idea is to switch those terms off for states, that already have K particles inside the segments, to which particles will be added. We have to take care to do this in a way that leaves the Hamiltonian hermitian on the level of second quantization, i.e. we have to adjust the terms $\widehat{h}_{\boldsymbol{ab}}$ and $\widehat{h}^{\dagger}_{\boldsymbol{ab}}$ into $\widehat{h}'_{\boldsymbol{ab}}$ and $(\widehat{h}^{\dagger}_{\boldsymbol{ab}})'$, such that $\widehat{h}'_{\boldsymbol{ab}} + (\widehat{h}^{\dagger}_{\boldsymbol{ab}})'$ is hermitian. For the $K = 2$ code of Figure 2.4, we can make the following adjustments:

$$c_i^{\dagger} c_j \;\mapsto\; \left(1 - \sum_{l,k<l \,\in\, \mathrm{B}} c_k^{\dagger} c_k c_l^{\dagger} c_l\right) c_i^{\dagger} c_j \left(1 - \sum_{w,v<w \,\in\, \mathrm{A}} c_v^{\dagger} c_v c_w^{\dagger} c_w\right) . \tag{2.99}$$

2.8 Notations

$[...]$	The set of integers from 1 to the argument.
$\hat{=}$	Relation used to express the correspondence between fermionic operators/states to qubit counterparts (according to some fermion-to-qubit mapping).
$\boldsymbol{a}$	Element of $[N]^{\otimes l}$. Length-l N-ary vector parametrizing orbitals in the Hamiltonian terms $\widehat{h}_{\boldsymbol{ab}}$.
$A^{(-1)}$	A $(N \times N)$ binary matrix defining a conventional encoding (decoding).
$\boldsymbol{b}$	Element of $\mathbb{Z}_2^{\otimes l}$. A length-l binary vector determining operator types in $\widehat{h}_{\boldsymbol{ab}}$.
$\mathcal{B}$	The basis of a space of fermions on N orbitals, which is possibly smaller than the Fock-space basis.
$c_j^{(\dagger)}$	An element of $\mathcal{L}(\mathcal{F}_N)$. A fermionic annihilation (creation) operator.
$(\mathbb{C}^2)^{\otimes n}$	The vector space of n-qubit states.
$\boldsymbol{d}$	A binary vector function $\mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2^{\otimes N}$. The decoding function.
$\boldsymbol{e}$	Binary vector function $\mathbb{Z}_2^{\otimes N} \mapsto \mathbb{Z}_2^{\otimes n}$. The encoding function.
$\boldsymbol{\varepsilon^q}$	A binary vector function $\mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2^{\otimes n}$. The update function which plays a role for nonlinear encodings, see (2.29).

$\mathcal{F}_N$	The fermionic Fock space restricted on N orbitals.
$F(j)$	A Subset of $[n]$ The flip set with respect to orbital j, see (2.12).
$\widehat{h}_{\boldsymbol{ab}}$	A fermion operator $\mathrm{span}(\mathcal{B}) \mapsto \mathrm{span}(\mathcal{B})$. A generic term in a fermionic Hamiltonian, see (2.10).
$\mathcal{H}_N^M$	The antisymmetrized Hilbert space of M indistinguishable fermions on N orbitals.
$\mathbb{I}$	The identity operator on arbitrary spaces.
K	An element of $[N]$ The targeted Hamming weight of the vectors encoded in a binary code.
l	The length of a sequence of fermionic operators in $\widehat{h}_{\boldsymbol{ab}}$.
L	The weight of a Pauli string.
$\mathcal{L}(...)$	Denotes linear operators on the argument vector space.
M	The total number of particles in a system of N orbitals.
n	The number of qubits.
N	The number of fermionic orbitals.
$\boldsymbol{\nu}$	An element of $\mathcal{V} \subseteq \mathbb{Z}_2^{\otimes N}$. The N-orbital occupation vector representing a fermionic basis state, see (2.17).
$\boldsymbol{\omega}$	An element of $\mathbb{Z}_2^{\otimes n}$. A binary vector representing a product state in the n-qubit basis, see (2.18).
$\boldsymbol{p}$	Binary vector function $\mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2^{\otimes N}$ The parity function,used for the parity operators.
$P(j)$	A subset of $[n]$. The parity set of orbital j, see (2.12).
$\mathcal{P}$	Set of single-qubit Pauli operators $\{X, Y, Z\}$.
$\boldsymbol{q}$	An element of $\mathbb{Z}_2^{\otimes N}$. Binary vector denoting the occupational change of a vector $\boldsymbol{\nu}$ by a term $\widehat{h}_{\boldsymbol{ab}}$.
R	A binary $(N \times N)$ matrix, where the lower triangle (excluding the diagonal) is filled with ones, see (2.13).
θ_{ij}	A function $[N]^{\otimes 2} \mapsto \mathbb{Z}_2$. The discrete version of the Heaviside function, see (2.13).
$\boldsymbol{u_j}$	A binary vector in $\mathbb{Z}_2^{\otimes N}$ or $\mathbb{Z}_2^{\otimes n}$. The j-th unit vector, in which just component j is one.

$U(j)$	A subset of $[n]$ The update set of orbital j, see (2.12).
$\mathcal{U}^{\boldsymbol{a}}$	A linear operator: $\mathcal{L}\left((\mathbb{C}^2)^{\otimes n}\right)$. Update operator with respect to an occupation of $\boldsymbol{a}$, see (2.31) and (2.32).
$\mathcal{V}$	A subset of $\mathbb{Z}_2^{\otimes N}$. The set of all allowed occupation vectors $\boldsymbol{\nu}$, implementing the basis $\mathcal{B}$, see (2.17).
$\mathrm{w_H}(\cdot)$	A function $\mathbb{Z}_2^{\otimes N} \mapsto [N] \cup \{0\}$. The Hamming weight of a binary vector, which is the sum of its components.
$\mathfrak{X}$	A map $\left(\mathbb{Z}_2^{\otimes n} \mapsto \mathbb{Z}_2\right) \mapsto \mathcal{L}\left((\mathbb{C}^2)^{\otimes n}\right)$. The extraction superoperator, which relates binary functions to quantum gates, see (2.19) - (2.27).
$\mathbb{Z}_2$	The set of binary digits: $\{0, 1\}$.

2.9 Further work

Tapering qubits off

There is another approach that can help reducing the number of qubits by exploiting symmetries. Let us consider that a qubit Hamiltonian has a set of stabilizers. As stabilizer conditions constrain degrees of freedom, they can be taken into account to eliminate an equivalent number of qubits. In fact, we find that per stabilizer condition eliminated one qubit can be tapered off, which means removing a number of qubits equivalent to the number of stabilizer generators. In [34], where this idea was developed, a quantum algorithm is devised to render the action of the problem Hamiltonian trivial on the qubits to be removed. However, in the wake of [43], we have developed a perhaps simpler method, with which qubits can be tapered off a Hamiltonian H, given only the generating set of stabilizers. While those stabilizers have to take the form of Pauli strings, we might have to simulate the system in the negative subspace of some of them, which effectively corresponds to those stabilizers being defined with a minus sign: $S \in \pm\{\mathbb{I}, X, Y, Z\}^{\otimes n}$. Note that those stabilizers might not be a product of natural symmetries. Our original motivation was to test logical Hamiltonians, as they appear in the next chapter. These are Hamiltonians of systems in which stabilizer conditions are defined as part of a quantum code, but like before we can regard the stabilizers as

$\tau\backslash\sigma$	X	Y	Z
$\mathbb{I}$	h	h	h
X	$\pm h\cdot p$	$\pm ih\cdot p$	h
Y	$\mp ih\cdot p$	$\pm h\cdot p$	$\pm ih\cdot p$
Z	h	h	$\pm h\cdot p$

Table 2.3. Removing one qubit from a Pauli string $\tau\otimes h$, by eliminating a stabilizer generator $\pm\sigma\otimes p$. Here p and h are Pauli strings on $n-1$ qubits, and σ, τ are Pauli operators on an isolated qubit. The entries of the table show the $(n-1)$-qubit equivalents of $\tau\otimes h$ for various instances of τ (rows) and σ (columns).

boundary conditions imposed on a physical system of spins. Testing the Hamiltonian spectra, and thus the code space, is therefore not possible unless we somehow include these conditions. At this point, qubit tapering can be used to eliminate all stabilizer conditions and turn a logical Hamiltonian into a physical one. The spectrum of the latter can then be matched with its expectation to verify the subspace.
We will now describe our procedure in detail. It relies on a routine that tapers off one qubit for each stabilizer generator so the following scheme is to be repeated until no stabilizers are left. In the beginning, we isolate one qubit on which the selected stabilizer acts non-trivially as $\sigma\in\{X,\,Y,\,Z\}$. Let us write the stabilizer as $S=\pm\sigma\otimes p$, where the first register denotes the isolated qubit, and the second register is comprised of the remaining qubits, on which the stabilizer acts as the Pauli string $\pm p$. We now replace all logical operators (and remaining stabilizers), $\tau\otimes h$, with the entries (corresponding to the concrete instances of τ and σ) in Table 2.3.

Obviously, the isolated qubit has been removed, and the stabilizer S is discarded. Let us prove this method. The idea is that by the stabilizer, a Pauli string $\tau\otimes h$ is re-expressed as either $\mathbb{I}\otimes h'$ or $\pi\otimes h''$, where $\pi\in\{X,\,Y,\,Z\}/\{\sigma\}$. In the table, we have have conventionally chosen to fixed $\pi=Z$ in cases $\sigma\in\{X,\,Y\}$ and $\pi=X$ for the case $\sigma=Z$. This is achieved by multiplying $\tau\otimes h$ with $\sigma\otimes p$ in case $\tau\notin\{\mathbb{I},\,\pi\}$. Since now each term is acting on the isolated qubit as either π or $\mathbb{I}$, it can be disregarded from the underlying eigenvalue problem, as it must be in the $\pm$-eigenstate of π. To save qubits, we just replace it with its eigenvalue ±1: $\mathbb{I}\otimes h'\mapsto\otimes h'$ and $\pi\otimes h''\mapsto\pm h''$. Note that the concrete choice between the two eigenvalues is irrelevant: the case in which the tapering is performed within the (-1) eigenspace produces a Hamilto-

nian $H \mapsto H^-$, that is related to its $(+1)$ counterpart, H^+, by the unitary transform: $H^+ = p\ H^- p$, which means H^+ and H^- are isospectral.

Term reduction

When applying the tapering, the number of Hamiltonian terms is not increased by the procedure outlined above. However, it can very well decrease. Two Hamiltonian terms, h^0 and h^1, can for instance be related by the condition $h^1 = \pm h^0 \cdot S$. After S is eliminated by tapering, the two terms will not stay distinct, but rather be added or subtracted according to the sign, meaning that at least one or even both terms vanish. Even without eliminating qubits, this feature can be utilized to reduce the number of Hamiltonian terms. This is particularly useful for logical Hamiltonians, since for transforms concatenated with quantum codes, the number of terms tends to proliferate.[2] The problem is that while many of those terms are equivalent up to the multiplication by stabilizers, however they are not identical, such that classical software cannot merge or cancel them. We have defined a classical routine to eliminate redundant terms while maintaining the code space. Like before, we begin by choosing a qubit on which the first stabilizer acts as σ^1, and by multiplication fix each logical term as well as the remaining stabilizer generators to act on it as $\pi^1 \neq \sigma^1$ or $\mathbb{I}$, the identity. However, while discarding the first stabilizer we are not going to remove the selected qubit – a procedure that is repeated for every stabilizers in the list of generators. Note that we always have to select a qubit that is distinct from the ones already fixed. For a total of r stabilizer generators, we end up with each logical Pauli string h transformed into $\lambda^h \otimes \eta^h$, where $\lambda^h \in \bigotimes_{i=1}^{r} \{\mathbb{I}, \pi^i\}$ is a fixed string on the r qubits selected, and η^h the 'free' remainder of h on the $n - r$ qubits left. When every Hamiltonian string h is brought into this form, redundant terms cancel or merge. However, the Hamiltonian strings probably had an optimized Pauli weight as h, that is likely to be not conserved in $\lambda^h \otimes \eta^h$. To keep the weight optimized, we perform the above procedure within a table, such that in the end each original string h is stored together with its fixed form. We have thus obtained a look-up table with which every remaining string $\lambda^h \otimes \eta^h$ of the Hamiltonian can be mapped back to h, retrieving its optimized weight.

[2] Unpublished observations.