A Practical Quantum Schur Transform

by
William M. Kirby

Professor Frederick W. Strauch, Advisor

A thesis submitted in partial fulfillment
of the requirements for the
Degree of Bachelor of Arts with Honors
in Physics

WILLIAMS COLLEGE
Williamstown, Massachusetts
February 15, 2018

Abstract

We describe an efficient quantum algorithm for the quantum Schur transform. The Schur transform is an operation on a quantum computer that maps the standard computational basis to a basis composed of irreducible representations of the unitary and symmetric groups. We simplify and extend the algorithm of Bacon, Chuang, and Harrow, and provide a new practical construction as well as sharp theoretical and practical analyses. Our algorithm decomposes the Schur transform on n qubits into $O\left(n^4 \log\left(\frac{n}{\epsilon}\right)\right)$ operators in the Clifford+T fault-tolerant gate set. We extend our qubit algorithm to decompose the Schur transform on n qudits of dimension d into $O\left(d^{1+p} n^{3d} \log^p\left(\frac{dn}{\epsilon}\right)\right)$ primitive operators from any universal gate set, for $p \approx 3.97$.

Acknowledgments

Many people have contributed to this project, either directly or indirectly, so I will attempt to give thanks where it is due. I am grateful to all my friends at Williams, especially my fellow Physics majors, and to all the professors whose classes have inspired me, for helping to create such a vibrant intellectual community. In particular, to my physics professors: I can honestly say that although physics classes can be difficult and at times stressful, I can look back on every physics class I have taken here and remember clearly how it made me into a better thinker and physicist. To Prof. Charlie Doret, my first research advisor: thank you for giving me such a fun and exciting introduction to physics research. To Prof. William Wootters, my research advisor for sophomore through junior year: thank you for being one of the most inspiring and influential people in all of my Williams experience, and for patiently teaching me how to do theoretical physics. Prof. Wootters was also the second reader for this thesis: thank you for your insightful comments and suggestions. To my girlfriend Anna: thank you for being so supportive and loving, even when I got abstracted and preoccupied with this project, and for making happiness my ground state. To my family: thank you for continuing to love me, support me, and inspire me all the time, as you have for my entire life.

I have two most important acknowledgements. First, to my advisor in this thesis, Prof. Frederick Strauch: thank you for being such a thoughtful mentor. You helped me convert my excitement into productivity, and guided my productivity towards the creation of a coherent piece of work. I think you have been really brilliantly good at finding a balance between helping me and giving me the space to figure things out myself. It has truly been an honor to work with you, and I hope we will continue to talk about physics together for many years to come.

Finally, a special thank you to my mom, Mary Mullen-Kirby. My mom gave me thirteen years of homeschooling, from kindergarten to twelfth grade, and I could not imagine a richer or more inspiring education than the one I received. She worked tirelessly to find the best possible books and materials for me, and spent thousands and thousands of hours teaching me, discussing books with me, taking me and my brother on field-trips, and giving me little intellectual nudges to investigate nearly anything I thought was interesting. I finished high school loving to learn and extremely excited to devote myself to it, with all the tools I needed to make the most of college. For all that, I credit my wonderful family, and my mom most of all. Thank you so much!

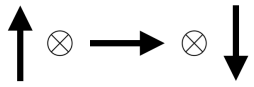
Executive Summary

We present an efficient quantum algorithm for the Schur transform on a quantum computer comprising n qudits. The Schur transform is a change of basis on the Hilbert space of the quantum computer, which maps the computational basis (composed of eigenvectors of the individual total spins and spin-projections) to a basis composed of states with a high degree of symmetry. This output basis is called the *Schur basis*, and it can be described in three equivalent ways:

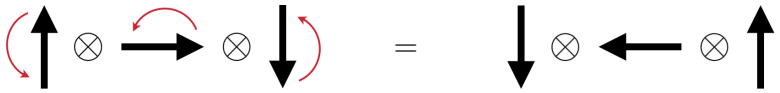
1. For the qubit case, the Schur basis is composed of eigenvectors of the total spin and spin projection of the whole register of qubits, which we call the *global spin*. For example, on two qubits, the Schur transform is the familiar Clebsch-Gordan transform, which maps the computational basis
$$|\uparrow\uparrow\rangle, \quad |\uparrow\downarrow\rangle, \quad |\downarrow\uparrow\rangle, \quad |\downarrow\downarrow\rangle \tag{1}$$
(for individual spin-projections $\uparrow \cong +\frac{1}{2}$ and $\downarrow \cong -\frac{1}{2}$) to the Schur basis
$$|\uparrow\uparrow\rangle, \quad \frac{1}{\sqrt{2}}\left(|\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle\right), \quad |\downarrow\downarrow\rangle, \quad \frac{1}{\sqrt{2}}\left(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle\right). \tag{2}$$
The Schur basis (2) is composed of eigenvectors of the global total spin j and spin-projection m: labeled as $|j, m\rangle$, the vectors in (2) are
$$|1,1\rangle, \quad |1,0\rangle, \quad |1,-1\rangle, \quad |0,0\rangle. \tag{3}$$
The Schur transform on two qubits is the matrix that performs this change of basis:
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} & 0 \end{pmatrix}. \tag{4}$$
Thus, one function of the Schur transform is to diagonalize the global total spin and spin-projection operators.

2. In the general qudit case, the Schur basis is a union of bases for irreducible representations of the *globally-symmetric unitary group*, the group of unitary operators on the Hilbert space of the quantum computer that can be written in the form $U^{\otimes n} \equiv \underbrace{U \otimes U \otimes \cdots \otimes U}_{n \text{ copies}}$, where U

is a unitary acting on one qudit; thus the globally-symmetric unitary group is isomorphic to the unitary group on one qudit. For example, given an initial state of three qubits

$|\psi\rangle =$  (5)

a globally-symmetric π-rotation (denoted U_π) about the "out-of-the-page" axis gives

$U_\pi|\psi\rangle \quad =$  (6)

(The red arrows represent the rotation.) Roughly speaking, in the Schur basis the globally-symmetric unitary operators decompose into blocks, which are the irreducible representations. *Irreducible* means that no other change of basis that can further decompose these blocks

3. Also in the general qudit case, the Schur basis is a union of bases for irreducible representations of the *symmetric group* acting on the qudit labels, i.e., the group of permutations of the qudits. As an example of the action of the symmetric group on the qudits, consider the same initial state of three qubits above (5): the result of the permutation $P_{(123)} = (123)$ (in cycle notation) is

$P_{(123)}|\psi\rangle \quad =$  (7)

(The blue arrows represent the permutation.)

Some applications of the Schur transform are decoherence-free encoding [2, 3, 4], quantum hypothesis testing [5, 6], spectrum estimation [7, 8], distortion-free universal entanglement concentration [9], and reference-frame independent quantum communication [10]. Bacon *et al.* [1, 11] were the first to describe the Schur transform as a quantum algorithm, in 2006, and Berg in 2012 described a generalization of their work [12]. However, none of these papers provide a simple construction for the algorithm or a tight analysis of its runtime.

In this thesis, we present an efficient quantum algorithm for the Schur transform on n qudits. Similar to the work of Bacon *et al.*, our algorithm is recursive, building up the total transform in a sequence of steps. However, we take advantage of the structure of each recursion step to obtain a much simpler construction (and tight analysis). By adding a logarithmic number of ancillary qudits to the register, we can perform a specific reordering of the basis after each recursion step. The reordering spaces the irreducible representations of the globally-symmetric unitary group (for the current iteration) evenly over the states of the ancillary qudits, so that the ancillary qudits index the representations. This allows the next recursion step to act "semi-locally" and independently on the distinct irreducible representations, whose individual dimensions are logarithmic in the

total dimension. This semi-local structure allows our algorithm to achieve polynomial instead of exponential runtime.

Our algorithm affords a simple analysis. For general qudits of dimension d, we decompose the Schur transform into a sequence of operators from any universal gate set, to accuracy ϵ (measured using a scaled trace norm; see [13]). The sequence length generated by our algorithm is

$$O\left(d^{1+p} n^{3d} \log^p\left(\frac{d\,n}{\epsilon}\right)\right) \tag{8}$$

where $p \approx 3.97$ (see chapter 5).

We further refine our analysis for the qubit case ($d = 2$), which is useful for common quantum computer implementations. For qudits or qubits, we first decompose the Schur transform exactly into two-level operations (operations that only act on two dimensions of the Hilbert space). For qubits, each of these can then be decomposed approximately using known methods (see appendix B) into

$$O\left(n \log\left(\frac{n}{\epsilon}\right)\right) \tag{9}$$

Clifford+T gates (a universal gate set that can be implemented fault-tolerantly [14], [15]), for an overall error bounded by ϵ. We will show that the sequence length of two-level operations for qubits is $O(n^3)$, which we multiply by (9) to obtain our Clifford+T gate sequence length of

$$O\left(n^4 \log\left(\frac{n}{\epsilon}\right)\right). \tag{10}$$

In fig. 1, we compare the actual two-level gate sequence lengths generated by our algorithm (for qubits) to the upper bound of $O(n^3)$.

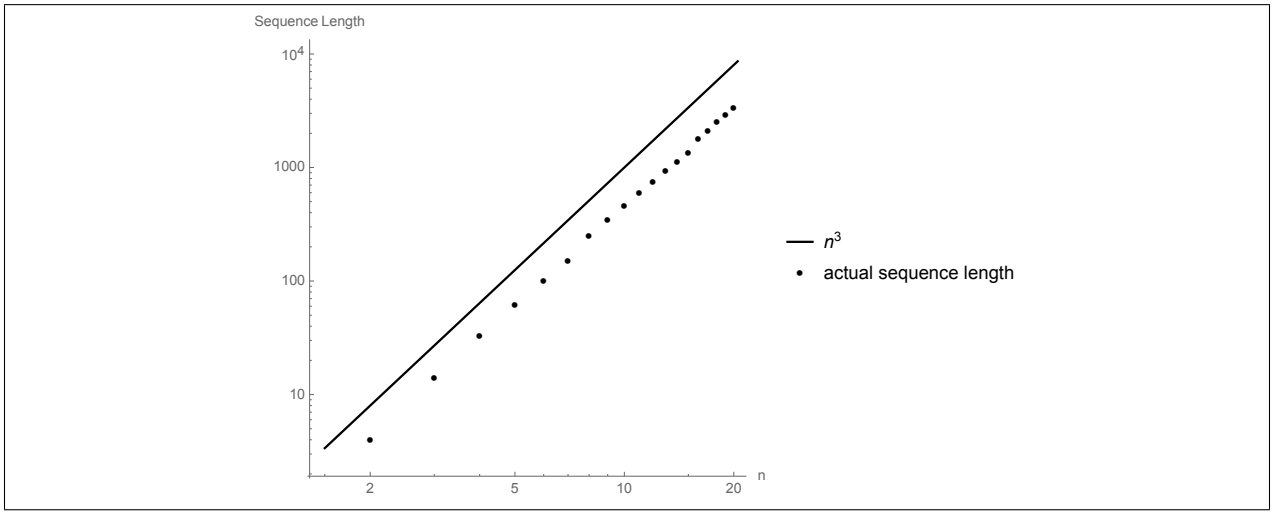


Figure 1: Two-level gate sequence lengths for n qubits

Contents

Chapter 1

Introduction

The study of quantum computing is roughly based on the following claim: certain types of computations that cannot be solved efficiently on a classical computer can be on a quantum computer. To assert that this is *the* fundamental tenet of the field would perhaps be an exagerration, but it is at least an approximately accurate statement of the main motivation for quantum computing. Two papers from the mid-1980s are often cited as marking the point at which quantum computing came into its own as a field: "Simulating Physics with Computers," by Richard Feynman [16], and "Quantum theory, the Church-Turing principle and the universal quantum computer," by David Deutsch [17]. These papers were among the first to suggest that quantum computing offered fundamentally different opportunities than classical computing. This claim is based on the observation that the dimension of a quantum system scales exponentially with the number of subsystems, so that by combining relatively few components we can create an enormous computational state space. Since 1985, when the Deutsch paper was published, the "quantum computing claim" has been validated by the discovery of efficient quantum algorithms for some tasks that lack efficient implementations for classical computers (for example, integer factorization [18], discrete function inversion [19], and the hidden subgroup problem [20]). However, the general picture of exactly which tasks are efficiently computable by quantum algorithms remains obscure. Nonetheless, and in part because some of the limits are still unknown, interest in quantum algorithm design continues to increase.

As the theory of quantum computing has developed, work on actually building such a computer has progressed in parallel. This is an extremely complex task, with difficulties ranging in character from fundamental theory to pure engineering. There are many potential means for realizing *qudits*, the basic quantum subsystems on which quantum computation can be performed, and the operations to be performed on them: researchers have built qudits out of trapped photons [21], trapped ions [22], superconducting circuits [23], and Nitrogen-vacancy centers in diamonds [24], among other systems, but to date the task of assembling a sufficient number of stable, operational qudits to perform significant quantum computation has proved elusive. However, there are indications that this goal may be reached within the next few years. Some recent results have been

very encouraging: In August of 2016, the Munroe group at Joint Quantum Institute, University of Maryland, demonstrated a universal, programmable 5-qubit quantum computer [25]. And in February of 2017, a team based at the University of Sussex, UK, published the first "blueprint" for a full-scale, modular, ion-trap quantum computer, and have begun construction of a prototype [26].

Thus, it appears that we are approaching a critical point in the development of quantum computing: the point at which the physical theory and the computational theory can merge. (In fact, one could argue that this point has already been reached: the Munroe group's 5-qubit computer has performed the quantum Fourier transform [27] – the principal subroutine in Shor's algorithm for integer factorization [18] – only it is still so small that a classical computer can outperform it by many orders of magnitude [25]). In any case, this is a hopeful time for the field as a whole, and in particular, it is an exciting time to be designing new quantum algorithms.

1.1 The Schur transform

In this document, we present a classical algorithm to compile an efficient quantum algorithm for the Schur transform. The Schur transform [1] is a fundamental operation that can be performed on a quantum computer. Like the quantum Fourier transform, it is a change of basis that can be qualitatively described as shifting the set of symmetries exhibited by the basis. There are three equivalent ways of stating the operation that the Schur transform performs; we will show that they are equivalent in §2.3 through §2.7.

- As a first description, consider the case of *qubits* (two-dimensional qudits). The Hilbert spaces of the individual qubit comprise the various spin-projection states in our particular dimension. The Hilbert space $\mathcal{H}$ of the whole computer is the tensor product of these subspaces. One natural way to build a basis for $\mathcal{H}$ is to take all possible tensor products of the basis elements of the subspaces, taking one from each subspace. This is often called the *computational basis*, and it is well-adapted to expressing operations on the individual qudits in the computer.

  However, for operations that act on the computer as a whole, the computational basis may not be the best choice. In particular, we can interact with the total spin and spin-projection of the whole register of qubits, but the computational basis vectors do not have definite total spins or spin projections, in general. So if we want to perform operations on the total spin of the register, it is advantageous to choose basis vectors that do have definite total spin. This is the change of basis that the Schur transform performs.

  For example, consider a simple quantum computer made of two qubits. If we label the individual state vectors $|0\rangle$ and $|1\rangle$, then the computational basis is given by

$$|00\rangle, \quad |01\rangle, \quad |10\rangle, \quad |11\rangle \tag{1.1}$$

Of these, only $|00\rangle$ and $|11\rangle$ have definite total spin. An alternative basis whose vectors do have definite total spin is

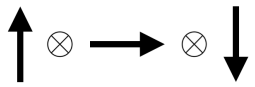
$$|00\rangle, \quad \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle), \quad |11\rangle, \quad \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \tag{1.2}$$

The first three states have total spin 1, and the last one has total spin 0. Their spin-projections, from left to right, are 1, 0, -1, and 0. A transformation between these bases is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} & 0 \end{pmatrix} \tag{1.3}$$

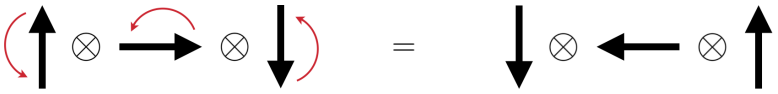
This is the simplest example of a Schur transform. When it acts on only two subsystems, the Schur transform is equivalent to the more familiar Clebsch-Gordan transform [29].

- There are two other ways to describe the Schur transform, which capture the picture for qudits of arbitrary dimension. The first is in terms of the unitary group $\mathcal{U}_d$. If we have n qudits with dimension d, we consider the group of operators $\mathcal{V} = \{U^{\otimes n} \mid U \in \mathcal{U}_d\}$, that is, unitary operators whose action on each qudit is the same. We refer to these as *globally-symmetric unitaries*. For example, given an initial state of three qubits

$|\psi\rangle =$ 

(1.4)

a globally-symmetric π-rotation (denoted U_π) about the "out-of-the-page" axis gives

$U_\pi|\psi\rangle \quad =$ 

(1.5)

It is not possible to simultaneously diagonalize every globally-symmetric unitary, but we can choose a basis such that the elements in $\mathcal{V}$ are "maximally-diagonal," in the sense that they are block-diagonal, and each block is as small as possible. The blocks are called *irreducible representations* (irreps) of the globally-symmetric unitary group, and they are identical to the subspaces of the Hilbert space that have definite values for the total spin (for example, the first three states in (1.3), which form a spin-1 subspace of the overall Hilbert space). We will see in Chapter 2 that this change of basis is the same as the one above.

- The other way to describe the Schur transform is in terms of the symmetric group $\mathcal{S}_n$ (the group of permutations of n objects). If we have n qudits, the symmetric group can act on the register by permuting the states of the qudits. For example, given the initial state of three

qubits above (1.4), the action of the permutation $P_{(123)} = (123)$ (in cycle notation) is

$P_{(123)}|\psi\rangle$ =  (1.6)

Any such permutation can be written as a unitary operator on the whole register, and while it is not possible to simultaneously diagonalize all such operators, it is possible to choose a a basis so that they are block-diagonal, with blocks as small as possible. We will see in Chapter 2 that this change of basis is also the same as the one above, up to a reordering of the qudits.

1.2 Applications of the Schur transform

Here are a few of the better known applications of the Schur transform:

- A *decoherence-free encoding* on a quantum computer is a means of encoding the computational qudits in a subspace of the overall register, such that the encoded states are not susceptible to uniform noise arising from decoherent coupling to a background field. This can be acheived by choosing the subspace to exploit the symmetry of the coupling ([2], [3], [4]), and one can show [1] that the Schur transform takes the computational basis to a basis that satisfies the conditions for a decoherence-free encoding.
- The Schur transform can be used to efficiently implement *quantum hypothesis testing* (determining which of two possible states an ensemble of identical particles occupies) ([5], [6]), as well as *spectrum estimation* (also for an ensemble of identical particles) ([7], [8]).
- *Entanglement concentration* is the process of producing a strongly-entangled state of some composite system from a collection of weakly-entangled states of identical systems. *Universal entanglement concentration* refers to the implementation of this process when the states of the initial systems are unknown, and when only local operations on the entangled subsystems are allowed. Universal entanglement concentration can be implemented by performing the Schur transform locally on each of the collections of subsystems, and measuring over the obtained symmetric group blocks: this implementation is in fact *distortion-free*, meaning that the resulting state is maximally-entangled [9].
- By a similar method can be used to implement *reference-frame independent quantum communication* [10]. This implementation is analogous to that used in decoherence-free encoding, since the lack of a shared reference frame can be thought of as applying an effective unknown unitary operator symmetrically to every qudit.

1.3 Previous work and overview of current project

In 2006, David Bacon, Isaac Chuang, and Aram Harrow published a short paper in Physical Review Letters, as well as a longer paper on *arxiv.org*, in which they qualitatively described a quantum algorithm for the Schur transform. To our knowledge, this was the first paper discussing a quantum algorithm for the Schur transform. A *quantum algorithm*, in this context, means a decomposition of the desired operation into a sequence of primitive gates from some universal gate set. *Efficient* means that the length of the sequence of primitives scales as a polynomial in n, the number of qudits in the register of the quantum computer. Bacon *et al.* do not explicitly find the exact "quantum runtime" (length of decomposition into primitive gates), nor do they provide an explicit description of their algorithm. A subsequent thesis [12] describes a generalization of Bacon *et al.*'s construction, but similarly does not provide an algorithm or an exact quantum runtime.

We originally began the project described herein by addressing (though not with the intention of limiting ourselves to) the task of building on Bacon *et al.*'s work by attempting to implement their algorithm and determine its quantum runtime more precisely. However, in beginning to explicitly build the algorithm, we discovered a new, simpler algorithm that adheres more closely to the mathematical structure of the Schur transform. This allows an elegant implementation and simple analysis of the quantum runtime: our algorithm returns a sequence of primitive gates whose length scales as $O(n^4 \log(n/\epsilon))$ for a register of n qubits (and is empirically $\Theta(n^3)$), and is bounded (though not tightly) by $O\left(d^{1+p} n^{2d+1} \log^p\left(\frac{d\,n}{\epsilon}\right)\right)$ for n qudits of dimension d ($p \approx 3.97$). The product of either sequence is the Schur transform, with error ϵ in the trace norm [13] scaled by the dimension.

Our algorithm shares its outermost layer of structure with Bacon *et al.*'s construction: it recursively decomposes the Schur transform into a succession of operators built out of Clebsch-Gordan transforms. In the implementation of these operators and in the structure of the recursion step itself, we differ from Bacon *et al.*'s approach: in these steps we made structural simplifications that take advantage of symmetries in the problem. The key insight that allowed these improvements goes roughly as follows:

At each large-scale step in the algorithm, we have a decomposition of the full Hilbert space into some set of subspaces, each of which is an irrep for the unitary group for some subset of the qudits in the register (i.e., has a definite value of total spin in the case of qubits). We will show herein that a careful *reordering* of these subspaces at each step makes a simple, explicitly-analyzable algorithm possible. The reordering also makes the structure of the algorithm more clearly reflect the structure of the underlying mathematical objects. To perform the reordering, we add a logarithmic number of ancillary qudits to the register. Then, roughly speaking, the reordering spaces the irreducible representations of the globally-symmetric unitary group (for the current iteration) evenly over the states of the ancillary qudits, so that the ancillary qudits act as indices. This allows the next recursion step to act "semi-locally" and independently on the distinct irreducible representations, whose individual dimensions are logarithmic in the total dimension. This allows us to achieve polynomial instead of exponential runtime. Thus, all of the hard work in our algorithm is encoded

in the orderings of the bases. The ordering of a basis does not affect the structure of the vector space it spans, so it is interesting that the ordering does turn out to determine the efficiency of our iteration to the next basis.

As a simple example of the reordering step, consider the Schur transform on two qubits again (1.3). As we discussed above, the Schur basis for two qubits is composed of a spin-1 subspace and a spin-0 subspace. These correspond to the irreducible representations of the globally-symmetric unitary group. So in our reordering, we want to index these subspaces by an ancillary qubit (since there are only two irreducible representations, one ancillary qubit is sufficient). We add the ancillary qubit (to the "top" of the register, i.e., the beginning of the tensor product), giving

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} & 0 \end{bmatrix} \tag{1.7}$$

We then reorder to obtain

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1/\sqrt{2} & -1/\sqrt{2} & 0 \end{bmatrix} \tag{1.8}$$

Thus, the spin-1 subspace (highlighted in red) is indexed by the value $\uparrow$ for the ancillary qubit, and the spin-0 subspace (highlighted in blue) is indexed by the value $\downarrow$ for the ancillary qubit. Note that although the value $\uparrow$ for the ancillary qubit labels four states (rows in (1.8)), only three of them are actually used, and of the states labelled by $\downarrow$, only one is used. The values of the ancillary qubit create "slots" for the total-spin subspaces, which need only be large enough to store the subspace with greatest total spin (highest dimension). The leftover states in each slot can be ignored.

In Chapter 2, we will review the mathematical background and structure of the Schur transform in detail. In Chapter 3, we will discuss the Clebsch-Gordan transform, showing how its mathematical and physical descriptions coincide, and how a sequence of Clebsch-Gordan transforms can be used to build the Schur transform. In Chapter 4, we will describe our algorithm for the Schur transform.

Chapter 2

Mathematical Background

This chapter provides an overview of the mathematics that is necessary for understanding and studying the Schur transform. For detailed treatment of the material in this chapter, see [30]. Some of the shorter proofs are included in the main body, but for longer proofs we either leave the proof to appendix A, or reference [30].

2.1 Representations

Definition 2.1.1 :
A *group* is a set equipped with a binary operation, typically denoted by multiplication. The operation satisfies the following four properties: for a group G...

1. For any $x, y \in G$, $xy \in G$.
2. There exists $\epsilon \in G$ (sometimes denoted $\mathbf{I}$) called the *identity element*, such that $\epsilon x = x\epsilon = x$ for any $x \in G$.
3. For any $x \in G$, there exists $x^{-1} \in G$ such that $xx^{-1} = x^{-1}x = \epsilon$.
4. For any $x, y, z \in G$, $(xy)z = x(yz)$, so that without ambiguity we can write xyz.

We will be concerned with two particular groups: the symmetric group and the unitary group.

Definition 2.1.2 :
The *symmetric group* on n elements, denoted $\mathcal{S}_n$, is the group of permutations of n objects. Elements in $\mathcal{S}_n$ can be written using a few possible notations:

- $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 4 & 5 & 3 \end{pmatrix}$ denotes the permutation that maps the first element to the second, the

second to the first, the third to the fourth, the fourth to the fifth, and the fifth to the third. This notation is called *two-line* notation.

- Since the first line in two-line notation is really just a convenience that does not contain any actual information, we can drop it to obtain *one-line* notation: the same permutation as above is given by $(2, 1, 4, 5, 3)$.
- The third notation is called cycle notation, which is most easily illustrated by an example: in cycle notation, $(12)(345)$ denotes the above permutation. Within each pair of parentheses (called a cycle), the ordering gives the permutations, with the last element being mapped back to the first (thus we could equivalently write $(21)(534)$, for example). An elementary proof in group theory shows that any permutation can be written in this form.

Definition 2.1.3 :
The *unitary group* $\mathcal{U}_d$ is the group of $d \times d$ unitary matrices. A matrix U is unitary if $U^{-1} = U^\dagger$.

- For example, $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}$ is unitary.
- It is not hard to show that a $d \times d$ matrix is unitary if and only if all of its columns (or rows) are orthonormal.

Definition 2.1.4 :
A *homomorphism* is a map from one group to another that preserves the group operation, i.e.,

$$F : G \to H \quad \text{is a homomorphism if} \quad \forall x, y \in G, \quad F(xy) = F(x)F(y) \tag{2.1}$$

The group of homomorphisms from G to H is denoted $Hom(G, H)$.

Definition 2.1.5 :
A *representation* of a group G is a homomorphism $R : G \to GL_d$, where GL_d is the complex general linear group of dimension d (the group of $d \times d$ nonsingular complex matrices).

Equivalently, we can define a representation as a homomorphism $R : G \to GL(V)$ for some vector space V, where $GL(V)$ is the set of linear maps from V to itself (endomorphisms). In this case, V is called a *G-module*, or just a *module*. When doing so will not lead to confusion, we will omit the homomorphism R and write $g\mathbf{v}$ to indicate the action of $R(g)$ on an element $\mathbf{v} \in V$.

Note: all the representations we will use are *regular representations*. A detailed treatment of regularity is not necessary for our discussion, but can be found in [30].

Definition 2.1.6 :
The *dimension* of a representation R is d if $R : G \to GL_d$, or $dim(V)$ if $R : G \to GL(V)$.

If $R(g)$ is unitary for all $g \in G$, we say that R is a unitary representation. Herein we consider only finite dimensional unitary representations. We will assume this without explicitly stating it in the rest of this document.

2.2 G-homomorphisms and Reducibility

Definition 2.2.1 :
For G-modules V and W, a *G-homomorphism* is a linear operator $M : V \to W$ that satisfies

$$M(g\mathbf{v}) = gM(\mathbf{v}) \tag{2.2}$$

for any $g \in G$ and any $\mathbf{v} \in V$. The group of G-homomorphisms from V to W is denoted $Hom_G(V, W)$.

Let R and S refer to the representations of G carried by V and W, respectively (so R maps elements in G to linear operators on V, and S maps elements in G to linear operators on W). We can define the action of any $g \in G$ on $Hom(V, W)$ (the group of linear transformations from V to W) by

$$gM = S(g^{-1})MR(g) \tag{2.3}$$

for any $M \in Hom(V, W)$. Then a G-homomorphism can be equivalently defined as an operator $M \in Hom(V, W)$ such that

$$gM = M \tag{2.4}$$

for all $g \in G$. That is, M is a G-homomorphism if for any $g \in G$ and any $\mathbf{v} \in V$,

$$MR(g)\mathbf{v} = S(g)M\mathbf{v} \tag{2.5}$$

We say that M *preserves the action of* G.

Definition 2.2.2 :
A *G-isomorphism* is an isomorphism of V and W that preserves the action of G, i.e., a G-homomorphism that is bijective.

Equivalently, a G-isomorphism is a G-homomorphism that is unitary. We say that V and W are *equivalent* (as G-modules) and write

$$V \overset{G}{\cong} W \tag{2.6}$$

if they are G-isomorphic, i.e., if any G-isomorphism between them exists. We do not use the term *equivalent* lightly here: if V and W are equivalent, then we are free to treat them identically. This will be important as we discuss reducibility below. A module can be a direct sum of several submodules (see below), some of which may be equivalent: we will often refer to these as "copies" of the same submodule, which means that they are all equivalent to that submodule.

Definition 2.2.3 :
Given a G-module V, a *submodule* W of V is a subspace of V that is invariant under G.

That is, a subspace W of V is a submodule if $g\mathbf{w} \in W$ for any $\mathbf{w} \in W$ and $g \in G$. Every module V has at least two submodules: V and $\{\mathbf{0}\}$, where $\mathbf{0}$ denotes the zero vector in V. These are called *trivial* submodules; any other submodules are called *nontrivial*.

Definition 2.2.4 :
A module with no nontrivial submodules is called *irreducible*. Otherwise, it is called *reducible*. For short, we will often refer to an irreducible module or its associated representation as an *irrep*. When it is not clear from context, we will state explicitly whether we are refering to an irreducible module or to the associated representation.

Suppose V is a reducible G-module. Then let W be a nontrivial submodule of V. Consider the coordinate system specified by the basis $\mathcal{V} = \{\mathbf{w}_1, ..., \mathbf{w}_f, \mathbf{v}_{f+1}, ..., \mathbf{v}_d\}$ for V, where $\{\mathbf{w}_1, ..., \mathbf{w}_f\}$ is a basis for W. Let $R : G \to GL_d$ be the matrix representation defined by this coordinate system on the module V. Since W is invariant under G, we have for any $g \in G$,

$$R(g) = \left(\begin{array}{c|c} A(g) & B(g) \\ \hline 0 & C(g) \end{array}\right) \tag{2.7}$$

where $A(g)$and $C(g)$ are square matrices. Note that $A(g)$ represents the action of g on W.

Definition 2.2.5 :
For vector spaces U and W, the *direct sum* $V = U \oplus W$ is the vector space of pairs $(\mathbf{u}, \mathbf{w})$ for $\mathbf{u} \in U$ and $\mathbf{w} \in W$.

Equivalently, given bases $\mathcal{U}$ for U and $\mathcal{W}$ for W, $V = U \oplus W$ is the vector space with basis $\mathcal{U} \cup \mathcal{W}$. By this definition, any vector $\mathbf{v} \in V$ can be written uniquely as $\mathbf{v} = \mathbf{u} + \mathbf{w}$ for $\mathbf{u} \in U$ and $\mathbf{w} \in W$.

Definition 2.2.6 :
Let V be a G-module. Then V is *completely reducible* if it contains disjoint, nontrivial submodules U and W such that

$$V = U \oplus W \tag{2.8}$$

If V is completely reducible as above, then for R the matrix representation carried by V and defined by the basis $\mathcal{U} \cup \mathcal{W}$,

$$R(g) = \left(\begin{array}{c|c} A(g) & 0 \\ \hline 0 & B(g) \end{array}\right) \tag{2.9}$$

for square matrices $A(g)$ and $B(g)$, since U and W are invariant under G. In this case, we write $R(g) = A(g) \oplus B(g)$.

Definition 2.2.7 :
A group G is *reductive* if every (regular) representation of G is either irreducible or completely reducible.

Theorem 2.2.1 (Maschke's theorem [31], [32]) **:**
Any finite group G is reductive.

Proof. See appendix A. □

Theorem 2.2.2 :
The special unitary group $\mathcal{SU}_d$ is reductive.

Proof. See [30] (Theorem 2.4.5). □

Theorem 2.2.3 (Isotypic decomposition) **:**
Let G be a reductive group. Then for any G-module V,

$$V \overset{G}{\cong} \bigoplus_{i=1}^{k} V^{(i)} \tag{2.10}$$

for some irreps $V^{(i)}$. Note that the $V^{(i)}$ need not all be inequivalent; we will discuss multiplicities shortly.

Proof. We induct on the dimension of V. Let $d = \dim(V)$.

Suppose $d = 1$. Then V itself is irreducible, so we are done. Now suppose $d > 1$, and suppose that the hypothesis (2.10) holds for any dimension less than d. If V is irreducible, then we are done. If V is reducible, then since G is a reductive group, V contains disjoint, nontrivial submodules U and W such that

$$V = U \oplus W \tag{2.11}$$

Since U and W are nontrivial submodules, $\dim(U) < d$ and $\dim(W) < d$, so by the inductive hypothesis, (2.10) holds for U and W,

$$U \overset{G}{\cong} \bigoplus_{i=1}^{j} U^{(i)} \tag{2.12}$$

$$W \overset{G}{\cong} \bigoplus_{i=1}^{k} W^{(i)} \tag{2.13}$$

for some irreps $U^{(i)}$ and $W^{(i)}$, and thus

$$V \overset{G}{\cong} \left(\bigoplus_{i=1}^{j} U^{(i)} \right) \oplus \left(\bigoplus_{i=1}^{k} W^{(i)} \right) \tag{2.14}$$

□

These theorems together prove the following corollary, upon which our whole course of study is based.

Corollary 2.2.3.1 :
Any module of the symmetric group $\mathcal{S}_n$ (since it is finite), or the unitary group $\mathcal{U}_d$, can be decomposed into a finite direct sum of irreducible submodules, as in (2.10).

We now prove a useful result (theorem 2.2.4) about G-isomorphic representations (after a few preliminaries):

Definition 2.2.8 :
For vector spaces V and W, and $M : V \to W$ a linear transformation,

$$ker(M) = \{\mathbf{v} \in V \mid M(\mathbf{v}) = \mathbf{0}_W\}, \tag{2.15}$$
where $\mathbf{0}_W$ is the zero-vector in W;
$$image(M) = \{\mathbf{w} \in W \mid \exists\ \mathbf{v} \in V \text{ such that } M(\mathbf{v}) = \mathbf{w}\} \tag{2.16}$$

Lemma 2.2.1 :
For G-modules V and W, and $M : V \to W$ a G-homomorphism,

1. $ker(M)$ is a submodule of V, and
2. $image(M)$ is a submodule of W.

Proof. From linear algebra, both $ker(M)$ and $image(M)$ are subspaces of their respective parent spaces. For any $\mathbf{v} \in ker(M)$,

$$M(g\mathbf{v}) = gM(\mathbf{v}) = g\mathbf{0} = \mathbf{0} \ \Rightarrow\ g\mathbf{v} \in ker(M) \tag{2.17}$$

where the first step follows because M is a G-homomorphism. Similarly, for any $\mathbf{w} \in image(M)$, there exists some $\mathbf{v} \in V$ such that $M(\mathbf{v}) = \mathbf{w}$. But since M is a G-homomorphism,

$$g\mathbf{w} = gM(\mathbf{v}) = M(g\mathbf{v}) \ \Rightarrow\ g\mathbf{w} \in image(M) \tag{2.18}$$

where the last step follows because $g\mathbf{v} \in V$, since V is a G-module. Thus, G preserves both $ker(M)$ and $image(M)$. □

Theorem 2.2.4 (Schur's Lemma) **:**
Any G-homomorphism between irreducible G-modules is either a G-isomorphism or the zero map.

Proof. Let V and W be irreducible G-modules, and let $M : V \to W$ be a G-homomorphism. Since V is irreducible, and $ker(M)$ is a submodule of V (by the above Lemma), $ker(M) = V$ or

$ker(M) = \{\mathbf{0}\}$. Similarly, $image(M) = W$ or $image(M) = \{\mathbf{0}\}$. If $ker(M) = V$, then M is the zero map, and $image(M) = \{\mathbf{0}\}$. If $ker(M) = \{\mathbf{0}\}$, then by linearity, M is an isomorphism, and $image(M) = W$. □

Corollary 2.2.4.1 :
Let V be an irreducible G-module, with $R : G \to GL(V)$ the corresponding irrep. Then the only operators in $GL(V)$ that commute with $R(g)$ for all $g \in G$ are operators of the form $c\mathbf{I}$, where $c \in \mathbb{C}$, and $\mathbf{I}$ is the identity operator.

Proof. See appendix A. □

Theorem 2.2.5 :
For G-modules V and W with V irreducible, the multiplicity of V in W is given by the dimension of the G-homomorphism space from V to W, i.e., by

$$\text{dim } Hom_G(V, W) \tag{2.19}$$

Proof. See appendix A. □

Definition 2.2.9 :
The *dual* of a vector space V is the set V^* of linear maps from V to $\mathbb{C}$.

V^* is isomorphic to V, and we can neatly encode the mapping from V to V^* by writing elements of V^* as *bra* vectors when elements of V are written as *ket* vectors. For a representation $R : G \to GL(V)$, we define the *dual representation* $R^* : G \to GL(V^*)$ by $R^*(g)\langle v| := \langle v|R(g^{-1})$.

Lemma 2.2.2 :
For representations $R_1 : G \to GL(V_1)$ and $R_2 : G \to GL(V_2)$,

$$Hom(V_1, V_2) \overset{G}{\cong} V_1^* \otimes V_2 \tag{2.20}$$

Proof. See appendix A. □

2.3 The Schur Transform

The *Schur transform* relates representations of the unitary group $\mathcal{U}_d$ of $d \times d$ unitary operators, and the symmetric group S_n of permutations on n objects. We will think of both groups as acting on a quantum register composed of n d-dimensional qudits: the symmetric group will permute the

qudits, and the unitary group will act on them individually. We denote the Hilbert space of the whole register

$$(\mathbb{C}^d)^{\otimes n} = \underbrace{\mathbb{C}^d \otimes \mathbb{C}^d \otimes \cdots \otimes \mathbb{C}^d}_{n \text{ copies}} \tag{2.21}$$

We consider the following pair of representations:

- $P : S_n \to GL\left((\mathbb{C}^d)^{\otimes n}\right)$, defined by

$$P(s)|\phi_1\rangle \otimes |\phi_2\rangle \otimes \cdots \otimes |\phi_n\rangle = |\phi_{s^{-1}(1)}\rangle \otimes |\phi_{s^{-1}(2)}\rangle \otimes \cdots \otimes |\phi_{s^{-1}(n)}\rangle \tag{2.22}$$

  where s is any permutation (in S_n), i.e., $P(s)$ permutes the component states according to s.

- $Q : \mathcal{U}_d \to GL\left((\mathbb{C}^d)^{\otimes n}\right)$, defined by

$$Q(U)|\phi_1\rangle \otimes |\phi_2\rangle \otimes \cdots \otimes |\phi_n\rangle = U|\phi_1\rangle \otimes U|\phi_2\rangle \otimes \cdots \otimes U|\phi_n\rangle \tag{2.23}$$

  for any $U \in \mathcal{U}_d$.

In words, P maps an element of S_n to the corresponding permutation of the n qudits, and Q maps an element U of $\mathcal{U}_d$ to the "globally-symmetric" unitary operator that applies U to each qudit.

Example. Suppose we have a system of three qubits, so that $n = 3$ and $d = 2$. Let $s = (123)$ in cycle notation: this means that s maps the first object to the second position, the second object to the third position, and the third object to the first position. Let $U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Suppose our qubits are in the state

$$|\phi_1\rangle \otimes |\phi_2\rangle \otimes |\phi_3\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.24}$$

Then

$$P(s)|\phi_1\rangle \otimes |\phi_2\rangle \otimes |\phi_3\rangle = |\phi_3\rangle \otimes |\phi_1\rangle \otimes |\phi_2\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \tag{2.25}$$

and

$$Q(U)|\phi_1\rangle \otimes |\phi_2\rangle \otimes |\phi_3\rangle = U|\phi_1\rangle \otimes U|\phi_2\rangle \otimes U|\phi_3\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.26}$$

According to corollary 2.2.3.1, we can decompose P and Q as

$$P(s) \overset{S_n}{\cong} \bigoplus_{\alpha} \bigoplus_{j=1}^{n_\alpha} p_\alpha(s) \tag{2.27}$$

$$Q(U) \overset{\mathcal{U}_d}{\cong} \bigoplus_{\beta} \bigoplus_{j=1}^{m_\beta} q_\beta(U) \tag{2.28}$$

where $p_\alpha(s)$ and $q_\beta(U)$ are sets of inequivalent irreps with multiplicities n_α and m_β (more on the multiplicities later). We write these expressions as equivalencies rather than equalities to recall the fact that some change of basis will in general be required (in $(\mathbb{C}^d)^{\otimes n}$).

Next we show that $P(s)$ and $Q(U)$ commute: for $U|\phi_i\rangle = |\psi_i\rangle$,

$$P(s)Q(U)\bigotimes_{i=1}^{n}|\phi_i\rangle = P(s)\bigotimes_{i=1}^{n}|\psi_i\rangle = \bigotimes_{i=1}^{n}|\psi_{s^{-1}(i)}\rangle = Q(U)\bigotimes_{i=1}^{n}|\phi_{s^{-1}(i)}\rangle = Q(U)P(s)\bigotimes_{i=1}^{n}|\phi_i\rangle \tag{2.29}$$

We can use the corollary to Schur's Lemma (corollary 2.2.4.1) to further simplify the situation. Since $P(s)$ and $Q(U)$ commute, their irreps must also commute, and corollary 2.2.4.1 states that the only linear operators that commute with an irrep are scalar multiples of the identity. Therefore, the action of any $p_\alpha(s)$ on any $q_\beta(U)$ must be the identity, and vice versa, since the only scalar multiple of the identity that is also unitary is the identity itself (up to a global phase, which we can ignore). Thus, in the joint action $P(s)Q(U)$, the permutations can only act to reorder copies of the same irrep of $Q(U)$, and vice versa, so we can write

$$Q(U)P(s) \overset{\mathcal{U}_d\times S_n}{\cong} \bigoplus_\alpha \bigoplus_\beta \bigoplus_{j=1}^{m_{\alpha,\beta}} p_\alpha(s)\otimes q_\beta(U) \tag{2.30}$$

where $m_{\alpha,\beta}$ is the multiplicity of the tensor product representation $p_\alpha(s)\otimes q_\beta(U)$. But we can simplify even further, though to do so we must first make things temporarily more complicated.

Definition 2.3.1 :
The *algebra* generated by a group is the vector space spanned by the elements of the group (over some field of scalars: we will assume that the scalars are elements of $\mathbb{C}$). A bilineal product is defined on this vector space: for a group G with elements $\mathbf{g}_k$ and algebra $\mathbf{G}$, if $\mathbf{v} = \sum_j v_j\mathbf{g}_j \in \mathbf{G}$ and $\mathbf{w} = \sum_k w_k\mathbf{g}_k \in \mathbf{G}$, then

$$\mathbf{vw} = \sum_{j,k} v_j w_k \mathbf{g}_j\mathbf{g}_k \tag{2.31}$$

Let $\mathbf{P}$ and $\mathbf{Q}$ be the algebras generated by the images of P and Q: that is,

$$\mathbf{P} = span\{P(s), s\in S_n\} \tag{2.32}$$

$$\mathbf{Q} = span\{Q(U), U\in \mathcal{U}_d\} \tag{2.33}$$

Elements of $\mathbf{P}$ commute with elements of $\mathbf{Q}$, which we can see by extending (2.29) linearly to $\mathbf{P}$ and $\mathbf{Q}$: we say that $\mathbf{P}$ and $\mathbf{Q}$ *centralize* each other. Thus we can again apply the corollary to Schur's Lemma (corollary 2.2.4.1) to obtain $m_{\alpha,\beta} = 1$ or 0 for all α,β. The reason that 0 is now a possibility, when 1 was the only possibility before, is that a 0 multiplicity simply indicates the absence of that particular pairing of irreps from the sum, to which the corollary does not apply. Thus, we can reduce our decomposition of $Q(U)P(s)$ to its fully simplified form:

$$Q(U)P(s) \overset{\mathcal{U}_d\times S_n}{\cong} \bigoplus_\lambda q_\lambda(U)\otimes p_\lambda(s) \tag{2.34}$$

This property – the fact that the representations $P(s)$ and $Q(U)$ can be simultaneously decomposed into their irreps – is known as *Schur duality*.

We are now in a position to define (abstractly for now) the Schur transform: the Schur transform is a unitary operator that performs a change of basis from the computational basis (formed by tensor products of the basis states for the individual qudits) to a basis in which (2.34) is an equality. That is, we define the Schur transform to be the unitary operator U_{Sch} such that

$$U_{Sch}Q(U)P(s)U_{Sch}^{\dagger} = \bigoplus_{\lambda} q_{\lambda}^{d}(U) \otimes p_{\lambda}(s) \quad \left(= \sum_{\lambda} |\lambda\rangle\langle\lambda| \otimes q_{\lambda}^{d}(U) \otimes p_{\lambda}(s) \right) \tag{2.35}$$

Note that we have left the indices λ that label the irreps undefined so far: we will begin the next section with a discussion of the labeling of the irreps, but this is not necessary for our big-picture understanding of the operation.

In summary, we have shown that for any $U \in \mathcal{U}_d$ and any $s \in S_n$, $Q(U)$ and $P(s)$ as defined in (2.22) and (2.23) are simultaneously reducible to block-diagonal form, where the blocks correspond to irreps of both. We define the Schur transform U_{Sch} to be the change of basis that performs this reduction.

It will also be useful for us to have a notation for the components of the Schur transform in terms of modules. The computational representation space can be written as $\left(\mathbb{C}^d\right)^{\otimes n}$: then we can write mathematical statement of Schur duality as

$$\left(\mathbb{C}^d\right)^{\otimes n} \overset{\mathcal{U}_d \times S_n}{\cong} \bigoplus_{\lambda} \mathcal{P}_{\lambda} \otimes \mathcal{Q}_{\lambda}^{d} \tag{2.36}$$

where $\mathcal{Q}_{\lambda}^{d}$ is an irreducible Q-module and $\mathcal{P}_{\lambda}$ is an irreducible P-module. Then the Schur transform maps the computational basis to a basis composed of basis elements for $\mathcal{Q}_{\lambda}^{d}$ and $\mathcal{P}_{\lambda}$: this will be a very useful way to think of it.

2.4 Partitions and Young diagrams

In this section, we introduce some mathematical notations that will be useful to us in analyzing the irreps of the symmetric and unitary groups. See [33] for a detailed introduction to these concepts.

A *partition* of an integer n is a set of positive integers arranged in weakly decreasing order whose sum is n. We use the notation $\lambda \vdash n$ to indicate that λ is a partition of n, and to explicitly define a partition we write $\lambda = (\lambda_1, \lambda_2, ..., \lambda_\ell)$, where λ_i is called the ith *part* of the partition. We will always assume that this notation is implicitly defined. Thus, if $\lambda \vdash n$, then

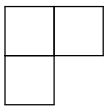
$$\sum_{i=1}^{\ell} \lambda_i = n \tag{2.37}$$

and

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_\ell \geq 0 \tag{2.38}$$

We call ℓ the *degree* of λ.

The use of λ to denote partitions as well as the labels of the irreps of the joint action of $Q(U)$ and $P(s)$ in (2.35) is not accidental. In fact, the partitions of n are exactly the labels for our irreps. In order to construct an irrep from a partition, we need to introduce a second notation for partitions: Young (or Ferrers) diagrams. The Young diagram of a partition λ is an array of boxes such that the ith row of the array contains λ_i boxes. For example, if $\lambda = (2, 1)$, then the Young diagram of λ is



(2.39)

A Young diagram labels an irrep, and a nice way to think of the Young diagram is as a sort of "template" for building vectors in the irrep. Let us see how this works, first for representations of the unitary group, and then for representations of the symmetric group:

2.5 A basis for $\mathcal{Q}_\lambda^d$

Definition 2.5.1 :
A *tableau* (plural: *tableaux*) of *shape* λ (we also call it a λ-*tableau*) is obtained by filling in the boxes in the Young diagram of λ with integers that we will call entries.

For our purposes, the entries in a λ-tableau will be labels for objects (either qubits in the case of $\mathcal{P}_\lambda$, or basis states in the case of $\mathcal{Q}_\lambda^d$), so if $\lambda \vdash n$, we will never need to use entries larger than n. For example, if we are working with 4 qubits, we could choose to label them 2, 11, 96, and 145, but it will be simpler if we simply label them 1, 2, 3, and 4. Here are a few exemplary $(3, 1)$-tableaux

$$\begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline 4 \\ \cline{1-1}\end{array}\,, \quad \begin{array}{|c|c|c|}\hline 2 & 2 & 2 \\ \hline 2 \\ \cline{1-1}\end{array}\,, \quad \begin{array}{|c|c|c|}\hline 4 & 4 & 3 \\ \hline 1 \\ \cline{1-1}\end{array} \tag{2.40}$$

Definition 2.5.2 :
A *standard tableau* is a λ-tableau whose entries are $\{1, 2, ..., n\}$ for $\lambda \vdash n$, and whose rows and columns strictly increase (from left to right, and top to bottom, respectively).

For example, the first tableau in (2.40) is a standard $(3, 1)$-tableau, but the other two are not. We should note that this is a stronger definition for a standard tableau than is normally used in representation theory: the condition that the entries form the set $\{1, 2, ..., n\}$ is normally not part of the definition. However, to obtain $\mathcal{Q}_\lambda^d$, we will only consider standard λ-tableaux whose entries

are $\{1, 2, ..., n\}$ for $\lambda \vdash n$, so it will be efficient for us to include that in the definition. The entries in these standard tableaux correspond to labels for the qudits in our system.

Definition 2.5.3 :
A *semistandard tableau* is a tableau whose entries are weakly increasing across rows and strictly increasing down columns.

Theorem 2.5.1 :
The dimension of $\mathcal{Q}_\lambda^d$ is equal to the number of semistandard λ-tableaux with entries in $\{1, 2, ..., d\}$.

Proof. See [30] (8.1.1 and 8.1.2). □

This result has the following useful corollary:

Corollary 2.5.1.1 :
For $\lambda \vdash n$ (for some integer n), if $\deg(\lambda) > d$, then $dim\left(\mathcal{Q}_\lambda^d\right) = 0$.

Proof. The degree of λ is the number of rows in the Young diagram of λ. Therefore, if $\deg(\lambda) > d$, there are not enough distinct elements in $\{1, 2, ..., d\}$ to fill the first column of λ in strictly increasing order, so no semistandard λ-tableaux with entries in $\{1, 2, ..., d\}$ exist. Thus by theorem 2.5.1, $dim\left(\mathcal{Q}_\lambda^d\right) = 0$. □

2.6 A basis for $\mathcal{P}_\lambda$

To build $\mathcal{P}_\lambda$, we again begin with λ-tableaux with entries $\{1, 2, ..., n\}$. The irreps of S_n are carried by modules known as the *Specht modules* [33], so $\mathcal{P}_\lambda$ is the Specht module of shape λ.

Theorem 2.6.1 :
For $\lambda \vdash n$, a basis for the Specht module $\mathcal{P}_\lambda$ is isomorphic to the set of standard λ-tableaux with entries $\{1, 2, ..., n\}$. Thus the dimension of $\mathcal{P}_\lambda$ is equal to the number of standard λ-tableaux with entries $\{1, 2, ..., n\}$.

Proof. See [33] (2.5). □

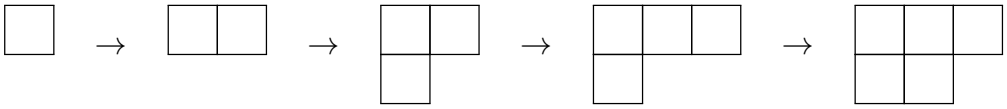
Luckily for us, this theorem has a nice physical interpretation. A useful property of any standard tableau is that if the largest entry is removed (with its box), the result is still a standard tableau (just now for a smaller value of n). If we reverse this argument, we can see that given any standard tableaux with entries $\{1, 2, ..., n\}$, if we add a new box containing the entry $n + 1$ in any position such that the new shape is still a partition, then we have a new standard tableau. Thus, a standard

tableau can be taken to represent a chain of partitions such that the first is a single box, and the Young diagram of each is the same as the Young diagram of the previous one, just with one new box. For example, the standard tableau

1	2	4
3	5	

(2.41)

can be taken to represent the following chain of Young diagrams (partitions):



(2.42)

Since the objects being permuted in our representation of interest ($\mathcal{P}_\lambda$) are the qudits, for our application the numbers $\{1, 2, ..., n\}$ are the qudit labels. So since a partition defines a Specht module, our interpretation of the standard tableaux as chains of partitions that "build up" λ one box at a time, starting from one box, means we can think of the basis for $\mathcal{P}_\lambda$ as made up chains of Specht modules, each of which has one qudit more than the previous one. But this gives us a very nice inductive method for building up the Specht modules: given all of the Specht modules for n qudits, to obtain the Specht modules for $n+1$ qudits we just need to add one box containing $n+1$ to all of the basis vectors (represented as standard tableaux) the Specht modules for n, and regroup the standard tableaux by their new shapes.

2.7 Schur duality and $\mathcal{P}_\lambda \otimes \mathcal{Q}_\lambda^d$

Now that we have bases for both $\mathcal{P}_\lambda$ and $\mathcal{Q}_\lambda^d$, we would like to find the form of vectors in $\mathcal{P}_\lambda \otimes \mathcal{Q}_\lambda^d$, applying Schur duality ((2.36)). Given some partition $\lambda \vdash n$, a basis vector in $\mathcal{P}_\lambda \otimes \mathcal{Q}_\lambda^d$ has abstract form

$$|T\rangle \otimes |t\rangle \tag{2.43}$$

where T is a standard λ-tableau with entries $\{1, 2, ..., n\}$ and t is a semistandard λ-tableau with entries taken from $\{1, 2, ..., d\}$. By "abstract form," we mean that there is some bijection from the vectors $|T\rangle$ to the basis vectors of $\mathcal{P}_\lambda$, and some bijection from the vectors $|t\rangle$ to the basis vectors in $\mathcal{Q}_\lambda^d$.

Suppose we want to decompose $\left(\mathbb{C}^d\right)^{\otimes n}$ into the irreps $\mathcal{Q}_\lambda^d$ of the unitary group, as we discussed in §2.3, and as illustrated in (2.28). We learned in §2.5 that each copy of $\mathcal{Q}_\lambda^d$ has a basis isomorphic to the set of semistandard λ-tableaux with entries in $\{1, 2, ..., d\}$: this is already reflected in (2.43). The critical observation here is that the copies of $\mathcal{Q}_\lambda^d$ are labeled by the set of standard λ-tableaux. Thus, before we even consider the symmetric group, we expect the form of the decomposition of $\left(\mathbb{C}^d\right)^{\otimes n}$ into irreps of the unitary group to be

$$\left(\mathbb{C}^d\right)^{\otimes n} \overset{\mathcal{U}_d}{\cong} \bigoplus_{\lambda \vdash n} \left(\bigoplus_T |T\rangle \otimes \mathcal{Q}_\lambda^d \right) \tag{2.44}$$

where the second direct sum is over standard tableaux T. Thus we can write vectors in the copy of $\mathcal{Q}_\lambda^d$ associated to a particular standard tableau T in the form (2.43). But as we stated above, this is the form we expect for the simultaneous decomposition of $\left(\mathbb{C}^d\right)^{\otimes n}$ into irreps of the unitary *and* symmetric groups. Therefore, as we hoped, decomposing $\left(\mathbb{C}^d\right)^{\otimes n}$ into the irreps $\mathcal{Q}_\lambda^d$ of the unitary group in this way simultaneously decomposes it into the irreps $\mathcal{P}_\lambda$ of the symmetric group, with each module acting as the multiplicity for the other so that when we consider the combined modules $\mathcal{P}_\lambda \otimes \mathcal{Q}_\lambda^d$, every possibility has a multiplicity of 1. Thus we have recovered Schur duality by a constructive approach, to complement our initial abstract argument. This remarkable structure is what makes the Schur transform possible (and useful).

2.8 Example

It is high time we give an example to illustrate these principles. Consider the case of two qubits: the corresponding Schur transform is

$$\textbf{Sch} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} & 0 \end{pmatrix} \tag{2.45}$$

Our claim is that the Schur basis is a basis for which (2.36) is an equality. Let us see how this works out:

λ is a partition of $n = 2$, so it takes two values: $\lambda = (2, 0)$ or $(1, 1)$. The standard and semistandard tableaux with shapes λ are thus

standard: $\begin{array}{|c|c|}\hline 1 & 2 \\ \hline \end{array}$, $\begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline \end{array}$ (2.46)

semistandard: $\begin{array}{|c|c|}\hline 1 & 1 \\ \hline \end{array}$, $\begin{array}{|c|c|}\hline 1 & 2 \\ \hline \end{array}$, $\begin{array}{|c|c|}\hline 2 & 2 \\ \hline \end{array}$, $\begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline \end{array}$ (2.47)

Thus by theorem 2.6.1, $\mathcal{P}_{(2,0)}$ and $\mathcal{P}_{(1,1)}$ are both one-dimensional, since there is one standard tableau with each shape. In fact, the Specht modules for this case are easily described. Recall that the elements of $\mathcal{S}_2$ are the permutations ϵ and (12), where ϵ represents the identity permutation. Then we have that $\mathcal{P}_{(2,0)}$ carries the *trivial representation*, which maps both permutations to the one-dimensional identity matrix (1), and $\mathcal{P}_{(1,1)}$ carries the *sign representation*, which maps ϵ to (1) and (12) to (-1).

For the unitary group, by theorem 2.5.1, $\mathcal{Q}^2_{(2,0)}$ is three-dimensional, since there are three semistandard tableaux with shape $(2, 0)$, and $\mathcal{Q}^2_{(1,1)}$ is one-dimensional, since there is one semistandard

tableau with shape $(1,1)$. $\mathcal{Q}^2_{(1,1)}$ is the trivial representation of $\mathcal{U}_d$. $\mathcal{Q}^2_{(2,0)}$ is more complex to describe in the abstract, so we will observe its structure by direct construction below.

By (2.36), we expect the dimension of $\mathcal{P}_\lambda$ to be the multiplicity of $\mathcal{Q}^2_\lambda$, and vice versa, for each λ. Thus, $\mathcal{P}_{(2,0)} \otimes \mathcal{Q}^2_{(2,0)}$ is 3-dimensional: it carries 3 copies of the trivial representation $\mathcal{P}_{(2,0)}$ and 1 copy of the unitary irrep $\mathcal{Q}^2_{(2,0)}$. Thus, $\mathcal{P}_{(1,1)} \otimes \mathcal{Q}^2_{(1,1)}$ is 1-dimensional: it carries 1 copy of the sign representation $\mathcal{P}_{(1,1)}$ and 1 copy of the trivial representation $\mathcal{Q}^2_{(1,1)}$. Let us check:

In the computational basis, the matrices for the permutations in $\mathcal{S}_2$ are

$$P(\epsilon) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad P((12)) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.48}$$

We transform to the Schur basis by applying the Schur transform (2.45) on the left and right:

$$\textbf{Sch}\ P(\epsilon)\ \textbf{Sch}^\dagger = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \textbf{Sch}\ P((12))\ \textbf{Sch}^\dagger = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{2.49}$$

Thus we see that, as expected, we have three copies of the trivial representation (highlighted in red) and one copy of the sign representation (highlighted in blue). Now let us consider a general qubit unitary written in the computational basis:

$$U = \begin{pmatrix} a & -b^* \\ b & a^* \end{pmatrix} \quad \text{for } |a|^2 + |b|^2 = 1 \tag{2.50}$$

Then in the computational basis, $Q(U) = U \otimes U$. We transform to the Schur basis by applying the Schur transform (2.45) on the left and right:

$$\textbf{Sch}\ Q(U)\ \textbf{Sch}^\dagger = \begin{bmatrix} a^2 & -\sqrt{2}ab^* & (b^*)^2 & 0 \\ \sqrt{2}ab & |a|^2 - |b|^2 & -\sqrt{2}a^*b^* & 0 \\ b^2 & \sqrt{2}a^*b & (a^*)^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.51}$$

So we see that, as expected, we have one copy of the three-dimensional irrep $\mathcal{Q}^2_{(2,0)}$ (highlighted in red) and one copy of the trivial representation $\mathcal{Q}^2_{(1,1)}$ (highlighted in blue). Thus, the Schur transform behaves exactly as we expect.

Chapter 3

The Clebsch-Gordan Transform

In this chapter we describe the Clebsch-Gordan (CG) transform, which is the operation that we will use to perform the recursion step in our construction of the Schur transform in the next chapter. In the first section we will describe the CG transform from an abstract mathematical standpoint, and in the second section we will describe the CG transform as it is more commonly known to physicists (for the qubit case). Subsequently, we will explain how the two pictures fit together, and in the next chapter we will show how the CG transform can be applied iteratively to construct the Schur transform.

3.1 The Clebsch-Gordan transform: Mathematics perspective

Let $\mathcal{Q}_\mu^d$ and $\mathcal{Q}_\nu^d$ be irreps of the unitary group acting on qudits. Consider the tensor product representation: $\mathcal{Q}_\mu^d \otimes \mathcal{Q}_\nu^d$. By corollary 2.2.3.1, we can decompose the tensor product itself into irreps: for $\mu \vdash n$ and $\nu \vdash m$,

$$\mathcal{Q}_\mu^d \otimes \mathcal{Q}_\nu^d \overset{\mathcal{U}_d}{\cong} \bigoplus_{\lambda \vdash (n+m)} \mathcal{M}_\lambda \otimes \mathcal{Q}_\lambda^d \tag{3.1}$$

where $\mathcal{M}_\lambda$ is the multiplicity space associated to the irrep $\mathcal{Q}_\lambda^d$. By theorem 2.2.5,

$$\mathcal{M}_\lambda \cong Hom(\mathcal{Q}_\lambda^d, \mathcal{Q}_\mu^d \otimes \mathcal{Q}_\nu^d) \tag{3.2}$$

where for simplicity of notation we have used $Hom(...)$ to denote a space of $\mathcal{U}_d$-homomorphisms (as opposed to the fully technical notation $Hom_{\mathcal{U}_d}(...)$; we will continue to use the simpler notation). The CG transform is the change of basis from the standard tensor product basis of $\mathcal{Q}_\mu^d \otimes \mathcal{Q}_\nu^d$ to the basis in which (3.1) is an equality [1].

In fact, for our purpose a slightly simpler case will be sufficient: we will restrict our attention to the case $\nu = (1)$, so that $\mathcal{Q}_\nu^d = \mathcal{Q}_{(1)}^d$ is the defining representation on a single qudit. This

case is particularly convenient because the multiplicities of the inequivalent irreps in its isotypic decomposition are all 1 [1].

Lemma 3.1.1 :
For $\mu \vdash n$ and $\lambda \vdash n+1$,

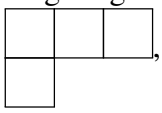
$$\dim Hom\left(\mathcal{Q}^d_\lambda, \mathcal{Q}^d_\mu \otimes \mathcal{Q}^d_{(1)}\right) = 1 \text{ or } 0 \tag{3.3}$$

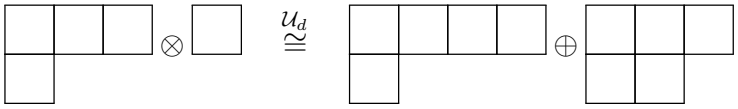
Further, the dimension is 1 iff $\lambda = \mu + \mathbf{e}_j$ for some j ($\mathbf{e}_j$ is the unit vector with a 1 at the jth index and 0s elsewhere: so $\mu + \mathbf{e}_j = (\mu_1, \mu_2, ..., \mu_j + 1, ..., \mu_\ell)$)

Proof. See [30] (8.1.1 and 8.1.2). □

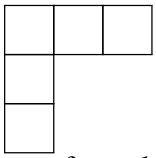
By lemma 3.1.1, we can rewrite (3.1) for the case $\nu = (1)$ as

$$\mathcal{Q}^d_\mu \otimes \mathcal{Q}^d_{(1)} \overset{\mathcal{U}_d}{\cong} \bigoplus_\lambda \mathcal{Q}^d_\lambda \tag{3.4}$$

where $\lambda = \mu + \mathbf{e}_j$ for some j such that λ is a valid partition and has degree less than or equal to d. We can visualize this schematically in terms of Young diagrams: for example, if $d = 2$ and $\mu = (3, 1)$, then the corresponding Young diagram is  , and (3.4) can be visualized as



(3.5)

If d were greater than 2, the Young diagram  would also appear on the right-hand side.

Thus we can think of the version of the CG transform that we are interested in as the "add-a-box" operation, which maps the standard tensor product basis for $\mathcal{Q}^d_\mu \otimes \mathcal{Q}^d_{(1)}$ to a basis in which (3.4) is an equality [1].

3.2 The Clebsch-Gordan transform: Physics perspective

So far, not just in the previous section, but in the majority of this treatise, we have been dwelling in a world much more mathematical than physical. The nearest we have approached a description of a physical object is in discussing qudits, but only in the sense of finite-dimensional Hilbert spaces. In this section, we will take one step closer to specificity and physicality by discussing spins and spin operators in the qubit case, understanding that as long as we remain tied to the mathematics it will ultimately make no difference how we describe our systems.

3.2.1 Review of Spin

Spin is a quantum system for which the eigenstates of the spin operators form a basis ([27]). We can write a spin eigenstate as $|j, m\rangle$, where j is a half-integer that parametrizes the total spin, and m is half-integer that parametrizes the projection of the spin along a particular axis, which we usually call the z-axis. j is greater than or equal to 0, and the magnitude of m is bounded by j, so in summary we have

$$j = 0, \frac{1}{2}, 1, \frac{3}{2}, 2, ... \tag{3.6}$$

$$m = -j, -j+1, -j+2, ..., j-2, j-1, j \tag{3.7}$$

The spin operators can be more fundamentally defined ([27]), but for our purposes here it will be sufficient to define them in terms of their eigenstates, which are parametrized by j and m.

Definition 3.2.1 :
The *total spin operator* (squared) $\mathbf{J}^2$ is defined by

$$\mathbf{J}^2|j, m\rangle = j(j+1)\hbar^2|j, m\rangle \tag{3.8}$$

Definition 3.2.2 :
The *spin projection operator* J^z is defined by

$$J^z|j, m\rangle = m\hbar|j, m\rangle \tag{3.9}$$

A particular spin system is defined by a particular value of j: for example, spin-$1/2$ refers to $j = 1/2$. Thus, a spin-j system has a basis made up of the eigenvectors of the spin-projection operator, which are indexed by the values of m: thus, a spin-j system is $(2j+1)$-dimensional [28].

In our construction of the CG transform, it will be valuable to have two more operators at our disposal: the *raising* and *lowering* operators. Qualitatively, the raising and lowering operators change the state of a system by increasing or decreasing (respectively) its spin-projection, without changing its total spin. They are defined as follows:

Definition 3.2.3 :
The *raising operator* J^+ is defined by

$$J^+|j, m\rangle = \sqrt{j(j+1) - m(m+1)}\hbar|j, m+1\rangle \tag{3.10}$$

Definition 3.2.4 :
The *lowering operator* J^- is defined by

$$J^-|j, m\rangle = \sqrt{j(j+1) - m(m-1)}\hbar|j, m-1\rangle \tag{3.11}$$

3.2.2 Composite Systems

Suppose we now have a system of two spins. One basis for the state space of this composite system is formed by the tensor products of the basis vectors of the subsystems. We call this the *computational basis*. For example, if our system is composed of two spin-$1/2$ particles, then the computational basis written out in full detail is

$$\begin{aligned} &|\frac{1}{2},\frac{1}{2}\rangle \otimes |\frac{1}{2},\frac{1}{2}\rangle \\ &|\frac{1}{2},\frac{1}{2}\rangle \otimes |\frac{1}{2},-\frac{1}{2}\rangle \\ &|\frac{1}{2},-\frac{1}{2}\rangle \otimes |\frac{1}{2},\frac{1}{2}\rangle \\ &|\frac{1}{2},-\frac{1}{2}\rangle \otimes |\frac{1}{2},-\frac{1}{2}\rangle \end{aligned} \tag{3.12}$$

where each vector has the form $|j_1, m_1\rangle \otimes |j_2, m_m\rangle$. However, these vectors are unnecessarily verbose: once we have stated that we have two spin-$(1/2)$ systems, we need not repeat the first $\frac{1}{2}$ in every vector, and we can omit the tensor product symbol as well. Thus, we can rewrite the same basis in the much cleaner form...

$$|\frac{1}{2},\frac{1}{2}\rangle, \quad |\frac{1}{2},-\frac{1}{2}\rangle, \quad |-\frac{1}{2},\frac{1}{2}\rangle, \quad |-\frac{1}{2},-\frac{1}{2}\rangle \tag{3.13}$$

where each vector has the form $|m_1, m_2\rangle$. One further notational simplification will give us the form that we will typically use *after* this section: this is to eliminate the annoying fractions entirely and simply index the substates by integers, so that for the states of a spin-j system would be labeled by the indices $0, 1, 2, ..., 2j$. This gives us a third form for our basis above:

$$|00\rangle, \quad |01\rangle, \quad |10\rangle, \quad |11\rangle \tag{3.14}$$

This third form will be the most convenient form to work with when we construct our algorithm for the Schur transform later, but for now we will use the second notation (3.13), where computational basis vectors are written in the form $|m_1, m_2\rangle$.

The computational basis is parametrized by the spins of the subsystems, which makes it very convenient if we want to operate on the subsystems independently. However, the composite system is itself a quantum system, and as such it has its own total spin and spin-projection, so as we have discussed above, we can transform to a basis parametrized by these quantities instead.

Before we proceed, we should define the composite spin operators. These are constructed in more or less the way one might guess: by adding the angular momentum operators. Thus if $\mathbf{J}_1^2$ and $\mathbf{J}_2^2$ are the total spin operators for the two subsystems, and J_1^z and J_2^z are the spin-projection operators, then the composite total spin and spin-projection operators are

$$\mathbf{J}^2 = (\mathbf{J}_1 + \mathbf{J}_2)^2, \qquad J^z = J_1^z + J_2^z \tag{3.15}$$

[29]. One can show ([27]) that for a system composed of a spin-j_1 and a spin-j_2, the composite total spin has eigenvalues

$$j = j_1 + j_2, j_1 + j_2 - 1, ..., |j_1 - j_2| \tag{3.16}$$

with the corresponding possible values of m.

We are now in a position to define the CG transform (for $d = 2$) from a physics perspective:

Definition 3.2.5 :
The *Clebsch-Gordan transform* on a system of two spins maps the computational basis to the basis composed of eigenvectors of the composite spin operators.

There are a number of ways of calculating the CG transform classically. Here we will present a simple method using the lowering operator (one could also use the raising operator). Eigenvectors of composite spin will be written in the form $|j, m\rangle$, while eigenvectors of individual spin-projections will be written in the form $|m_1, m_2\rangle$, as above. In the algorithm that follows, we will omit the variable names only for computational basis states (for example, $|\frac{1}{2}, \frac{1}{2}\rangle \equiv |m_1 = \frac{1}{2}, m_2 = \frac{1}{2}\rangle$).

1. We begin by finding the vector $|j, m\rangle = |j_1 + j_2, j_1 + j_2\rangle$, i.e., the vector for which both j and m have their maximum values.
2. For our running example, the only vector with $m = \frac{1}{2} + \frac{1}{2} = 1$ is $|m_1, m_2\rangle = |\frac{1}{2}, \frac{1}{2}\rangle$, since

$$J^z|\frac{1}{2}, \frac{1}{2}\rangle = (J_1^z + J_2^z)|\frac{1}{2}, \frac{1}{2}\rangle = \frac{1}{2}\hbar|\frac{1}{2}, \frac{1}{2}\rangle + \frac{1}{2}\hbar|\frac{1}{2}, \frac{1}{2}\rangle = \hbar|\frac{1}{2}, \frac{1}{2}\rangle \tag{3.17}$$

   so we have

$$|j = 1, m = 1\rangle = |\frac{1}{2}, \frac{1}{2}\rangle \tag{3.18}$$

3. Now we apply the composite lowering operator (which is simply the sum of the individual lowering operators) repeatedly, to obtain the rest of the eigenstates of composite spin-projection that have the same composite total spin.
4. For our running example, this looks like

$$\begin{aligned} |j = 1, m = 0\rangle &= J^-|j = 1, m = 1\rangle = (J_1^- + J_2^-)|\frac{1}{2}, \frac{1}{2}\rangle \\ &\rightarrow \frac{1}{\sqrt{2}}\left(|-\frac{1}{2}, \frac{1}{2}\rangle + |\frac{1}{2}, -\frac{1}{2}\rangle\right) \end{aligned} \tag{3.19}$$

$$\begin{aligned} |j = 1, m = -1\rangle &= J^-|j = 1, m = 0\rangle = (J_1^- + J_2^-)\frac{1}{\sqrt{2}}\left(|-\frac{1}{2}, \frac{1}{2}\rangle + |\frac{1}{2}, -\frac{1}{2}\rangle\right) \\ &\rightarrow 0\hbar|-\frac{3}{2}, \frac{1}{2}\rangle + \hbar|-\frac{1}{2}, -\frac{1}{2}\rangle + \hbar|-\frac{1}{2}, -\frac{1}{2}\rangle + 0\hbar|\frac{1}{2}, -\frac{3}{2}\rangle \\ &\rightarrow |-\frac{1}{2}, -\frac{1}{2}\rangle \end{aligned} \tag{3.20}$$

Note that in general we need to renormalize the resulting states at each step.

5. When the state $|j, -j\rangle$ is reached for the current value of j, find the state with the next lower value of j and the maximum possible value of m for that j. We can do this by finding a normalized linear combination of computational basis vectors that have the current value of m, but which is orthogonal to all linear combinations of those basis vectors that we have already found. This may not be unique in general: there may be larger multiplicities, in which case this and the following steps need to be repeated accordingly.

6. For our running example, we now want to find the state $|j = 0, m = 0\rangle$. The computational basis vectors with $m = 0$ are $|\frac{1}{2}, -\frac{1}{2}\rangle$ and $|-\frac{1}{2}, \frac{1}{2}\rangle$, so we want a normalized linear combination of these that is orthogonal to the linear combination of these basis vectors that we have already found, namely
$$\frac{1}{\sqrt{2}}\left(|\frac{1}{2}, -\frac{1}{2}\rangle + |-\frac{1}{2}, \frac{1}{2}\rangle\right) \tag{3.21}$$
In this case we can find the desired vector by inspection, but we could also solve for it systematically in the general case: here we obtain
$$|j = 0, m = 0\rangle = \frac{1}{\sqrt{2}}\left(|\frac{1}{2}, -\frac{1}{2}\rangle - |-\frac{1}{2}, \frac{1}{2}\rangle\right) \tag{3.22}$$

This method is important, so let us go through another example: a system composed of a spin-1 particle and a spin-$1/2$ particle.

1. The eigenvalues we expect for j are $1 + 1/2 = 3/2$ and $1 - 1/2 = 1/2$.

2. First find the state $|j = \frac{3}{2}, m = \frac{3}{2}\rangle$: the only computational basis state with $m = \frac{3}{2}$ is $\boxed{|1, \frac{1}{2}\rangle}$, so this is it.

3. Apply the composite lowering operator iteratively (see (3.11)):
$$\begin{aligned} |j = \frac{3}{2}, m = \frac{1}{2}\rangle &= (J_1^- + J_2^-)|1, \frac{1}{2}\rangle = \sqrt{2}\hbar|0, \frac{1}{2}\rangle + \hbar|1, -\frac{1}{2}\rangle \\ &\to \boxed{\sqrt{\frac{2}{3}}|0, \frac{1}{2}\rangle + \sqrt{\frac{1}{3}}|1, -\frac{1}{2}\rangle} \end{aligned} \tag{3.23}$$

$$\begin{aligned} |j = \frac{3}{2}, m = -\frac{1}{2}\rangle &= (J_1^- + J_2^-)\left(\sqrt{\frac{2}{3}}|0, \frac{1}{2}\rangle + \sqrt{\frac{1}{3}}|1, -\frac{1}{2}\rangle\right) \\ &= \frac{2}{\sqrt{3}}\hbar|-1, \frac{1}{2}\rangle + \sqrt{\frac{2}{3}}\hbar|0, -\frac{1}{2}\rangle + \sqrt{\frac{2}{3}}\hbar|0, -\frac{1}{2}\rangle + 0\hbar|1, -\frac{3}{2}\rangle \\ &\to \boxed{\sqrt{\frac{1}{3}}|-1, \frac{1}{2}\rangle + \sqrt{\frac{2}{3}}|0, -\frac{1}{2}\rangle} \end{aligned} \tag{3.24}$$

$$|j=\frac{3}{2},m=-\frac{3}{2}\rangle = (J_1^- + J_2^-)\left(\sqrt{\frac{1}{3}}|-1,\frac{1}{2}\rangle + \sqrt{\frac{2}{3}}|0,-\frac{1}{2}\rangle\right)$$
$$= 0\hbar|-2,\frac{1}{2}\rangle + \sqrt{\frac{1}{3}}\hbar|-1,-\frac{1}{2}\rangle + \frac{2}{\sqrt{3}}\hbar|-1,-\frac{1}{2}\rangle + 0\hbar|0,-\frac{3}{2}\rangle$$
$$\rightarrow \boxed{|-1,-\frac{1}{2}\rangle} \tag{3.25}$$

4. Now we want to find the state $|j=\frac{1}{2},m=\frac{1}{2}\rangle$: we know that it must be built of the computational basis states $|0,\frac{1}{2}\rangle$ and $|1,-\frac{1}{2}\rangle$ since these are the computational basis states with $m=1/2$. We also require that it be orthogonal to our already-found linear combination of these states:
$$|j=\frac{3}{2},m=\frac{1}{2}\rangle = \sqrt{\frac{2}{3}}|0,\frac{1}{2}\rangle + \sqrt{\frac{1}{3}}|1,-\frac{1}{2}\rangle \tag{3.26}$$
Thus we obtain the state
$$|j=\frac{1}{2},m=\frac{1}{2}\rangle = \boxed{\sqrt{\frac{1}{3}}|0,\frac{1}{2}\rangle - \sqrt{\frac{2}{3}}|1,-\frac{1}{2}\rangle} \tag{3.27}$$

5. Lastly, apply the composite lowering operator to the $|j=\frac{1}{2},m=\frac{1}{2}\rangle$ state to obtain the final state we desire:
$$|j=\frac{1}{2},m=-\frac{1}{2}\rangle = (J_1^- + J_2^-)\left(\sqrt{\frac{1}{3}}|0,\frac{1}{2}\rangle - \sqrt{\frac{2}{3}}|1,-\frac{1}{2}\rangle\right)$$
$$= \sqrt{\frac{2}{3}}\hbar|-1,\frac{1}{2}\rangle + \sqrt{\frac{1}{3}}\hbar|0,-\frac{1}{2}\rangle - \frac{2}{\sqrt{3}}\hbar|0,-\frac{1}{2}\rangle - 0\hbar|1,-\frac{3}{2}\rangle$$
$$= \sqrt{\frac{2}{3}}\hbar|-1,\frac{1}{2}\rangle - \sqrt{\frac{1}{3}}\hbar|0,-\frac{1}{2}\rangle$$
$$\rightarrow \boxed{\sqrt{\frac{1}{3}}|0,-\frac{1}{2}\rangle - \sqrt{\frac{2}{3}}|-1,\frac{1}{2}\rangle} \tag{3.28}$$

Having found the eigenvectors of the composite spins, writing down the CG transform is simply a matter of choosing an ordering for our mapping from the computational basis to the composite spin eigenvector basis. For both of our examples above, here are the results written as matrices, where we have ordered the composite spin eigenvector basis by values of m, which gives block diagonal form for our matrices:

$j_1 = \frac{1}{2}, j_2 = \frac{1}{2}$

$$\text{CG transform: } \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \end{pmatrix} \tag{3.29}$$

$$\text{for input in the form: } \begin{pmatrix} |\frac{1}{2}, \frac{1}{2}\rangle \\ |\frac{1}{2}, -\frac{1}{2}\rangle \\ |-\frac{1}{2}, \frac{1}{2}\rangle \\ |-\frac{1}{2}, -\frac{1}{2}\rangle \end{pmatrix} \quad \text{and output in the form: } \begin{pmatrix} |j=1, m=1\rangle \\ |j=1, m=0\rangle \\ |j=-1, m=-1\rangle \\ |j=0, m=0\rangle \end{pmatrix}$$

$j_1 = 1, j_2 = \frac{1}{2}$

$$\text{CG transform: } \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{\frac{1}{3}} & \sqrt{\frac{2}{3}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{\frac{2}{3}} & \sqrt{\frac{1}{3}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & -\sqrt{\frac{2}{3}} & \sqrt{\frac{1}{3}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{\frac{1}{3}} & -\sqrt{\frac{2}{3}} & 0 \end{pmatrix} \tag{3.30}$$

$$\text{for input in the form: } \begin{pmatrix} |1, \frac{1}{2}\rangle \\ |1, -\frac{1}{2}\rangle \\ |0, \frac{1}{2}\rangle \\ |0, -\frac{1}{2}\rangle \\ |-1, \frac{1}{2}\rangle \\ |-1, -\frac{1}{2}\rangle \end{pmatrix} \quad \text{and output in the form: } \begin{pmatrix} |j=\frac{3}{2}, m=\frac{3}{2}\rangle \\ |j=\frac{3}{2}, m=\frac{1}{2}\rangle \\ |j=\frac{3}{2}, m=-\frac{1}{2}\rangle \\ |j=\frac{3}{2}, m=-\frac{3}{2}\rangle \\ |j=\frac{1}{2}, m=\frac{1}{2}\rangle \\ |j=\frac{1}{2}, m=-\frac{1}{2}\rangle \end{pmatrix}$$

This is one method for constructing the CG transform as it is commonly known to physicists: other methods exist (for example, [34], [35]). In addition, generalizations of the above construction exist for the case of qudits ([**?**], for example). For our purposes it is sufficient to know that the CG transforms are classically calculable, since in our algorithm that part of the computation will be classical.

3.2.3 Putting the pieces together

We would like to reconcile the two descriptions we have given for the CG transform in the last two sections, starting with the case $d = 2$. We begin by noting the immediate similarities: namely...

1. In the mathematical description of the CG transform, we specialized to decomposing a tensor product of $\mathcal{Q}_\mu^d$ (for some $\mu \vdash n$) and $\mathcal{Q}_{(1)}^d$ into irreps. We can think of each of the irreps $\mathcal{Q}_\mu^d$ and $\mathcal{Q}_{(1)}^d$ as subsystems to be combined into a larger composite system.
2. The physical description of the CG transform is also based on the concept of combining two subsystems into a larger composite system, with the only qualitative differences being that both of the subsystems are assumed to be definite spins, and that the spins are allowed to be different (recall that in the mathematical description we assumed that all of the boxes had entries in $1, 2, ..., d$ for the semistandard tableaux).

As it turns out, the generality of the physical CG transform in the dimensions of the subsystems and the generality of the mathematical CG transform in the first irrep in the tensor product are intimately related. This is a consequence of the powerful fact that from a quantum informational perspective, two systems with the same dimension are equivalent as long as they carry the same representation of $\mathcal{U}_d$ (in the current case, $\mathcal{U}_2$). We take advantage of this fact whenever we talk about a qudit in the abstract: the structure is the same whether the physical qudit is an ion, a photon, or a superconducting quantum circuit. In our case, the equivalence is between an irrep (in the mathematical picture) and a spin (in the physical picture). Since both are perfectly legitimate quantum systems in their own rights, as long as they have the same dimensions and carry the same representation of $\mathcal{U}_2$ they can be treated as equivalent.

The operation we will want to perform with our CG transforms is the addition of a single qudit to a larger register of qudits, which is why we constrained the second irrep in our mathematical description to be $\mathcal{Q}_{(1)}^d$. The other qudits in the register will have previously been decomposed into their irreps by a Schur transform, so we can consider their irreps separately: hence the generality of the partition μ that labels the first irrep in our mathematical description (3.4). Any particular $\mathcal{Q}_\mu^d$ is just a Hilbert space whose dimension is determined by the number of semistandard μ-tableaux with entries in $1, 2, ..., d$ (as given in (**??**)). This is why, for example, it is correct to say that a composite system of two spin-$1/2$ particles decomposes into a spin-1 and a spin-0: the three spin-1 states correspond to the irrep $\mathcal{Q}_{(2)}^2$, and the singlet state, with spin-0, corresponds to the irrep $\mathcal{Q}_{(1,1)}^2$ (see fig. 3.1).

We should not interpret the correspondences in fig. 3.1 as direct identifications between the state vectors and tableaux, but the spanned subspaces are correct. The main point is that we can now perform, for example, the CG transform on $\mathcal{Q}_{(2)}^2 \otimes \mathcal{Q}_{(1)}^2$ (i.e., add another qubit), by treating the vector space $\mathcal{Q}_{(2)}^2$ as if it were a spin-1 particle and using the physicists' CG transform. Thus, as we will discuss in the next chapter, we can use a direct sum of physicists' CG transforms, each of which acts on two systems, to perform a "super-CG transform" that adds one qudit to a register of qudits that have already been decomposed into irreps: this will be the recursion step in our implementation of the Schur transform.

Definition 3.2.6 :
We will write *S-CG*$_k$ to denote the super-CG transform that adds a single qudit to k qudits that

Triplets (spin-1) $= \mathcal{Q}^2_{(2)}$:

$$\left\{ \begin{array}{c} |m=1\rangle = |\frac{1}{2},\frac{1}{2}\rangle \\ |m=0\rangle = \frac{1}{\sqrt{2}}\left(|\frac{1}{2},-\frac{1}{2}\rangle + |-\frac{1}{2},\frac{1}{2}\rangle\right) \\ |m=-1\rangle = |-\frac{1}{2},-\frac{1}{2}\rangle \end{array} \right\} \quad \Longleftrightarrow \quad \left\{ \begin{array}{|c|c|} \hline 1 & 1 \\ \hline \end{array} \; , \; \begin{array}{|c|c|} \hline 1 & 2 \\ \hline \end{array} \; , \; \begin{array}{|c|c|} \hline 2 & 2 \\ \hline \end{array} \right\}$$

Singlet (spin-0) $= \mathcal{Q}^2_{(1,1)}$:

$$\left\{ |m=0\rangle = \frac{1}{\sqrt{2}}\left(|\frac{1}{2},-\frac{1}{2}\rangle - |-\frac{1}{2},\frac{1}{2}\rangle\right) \right\} \quad \Longleftrightarrow \quad \left\{ \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline \end{array} \right\}$$

Figure 3.1: CG (Schur) transform on two qubits

have already been decomposed into their irreps.

Chapter 4

The Schur Transform

In this chapter we will describe the abstract structure of a quantum algorithm that performs the Schur transform on n qudits. We will then prove that this algorithm is efficient for *qubits* ($d = 2$; in the conclusion we will show that it is also efficient for qudits). In particular, we will show that the qubit algorithm can be decomposed into a sequence of

$$O(n^3) \tag{4.1}$$

two-level gates. A two-level gate is unitary operator that only acts on two dimensions, i.e., is isomorphic to a 2×2 unitary. Each two-level gate can be decomposed using known methods (see appendix B) into $O(n \log(1/\epsilon))$ gates from the Clifford+T set, which can be implemented fault-tolerantly [14], [15]. Here ϵ is the error, calculated using a scaled trace norm (also known as the Frobenius norm; see [13]); for unitaries U and V of dimension d, the error is given by

$$\epsilon(U, V) = \sqrt{\frac{1}{d}\text{Tr}\left[(U - V)^\dagger (U - V)\right]} \tag{4.2}$$

We will describe the qubit algorithm in detail (pseudocode) and give examples and two-level gate sequence lengths.

Our algorithm for the Schur transform will be recursive, using the CG transform as its recursion step. This outermost layer of structure is the same as that used by Bacon, Chuang, and Harrow in [1] and [11], but that is as far as the resemblence goes: our implementation is distinct and offers advantages in simplicity and analysis.

4.1 Abstract implementation

Our construction of the Schur transform relies on the following argument (similar but not identical to the argument in [1]):

- By Schur duality (2.36), if we decompose a register of n qudits with dimension d into irreps of the unitary group, i.e., if we decompose $\left(\mathbb{C}^d\right)^{\otimes n}$ into a direct sum of irreps $\mathcal{Q}^d_\lambda$ of $\mathcal{U}_d$, then we will have simultaneously decomposed it into irreps $\mathcal{P}_\lambda$ of the symmetric group. The irreps of the unitary group act as the multiplicity spaces for the irreps of the symmetric group, and vice versa (see §2.7).
- The CG transform (in the form that we can implement using the physicists' description as discussed in §3.2 and §3.3) acts on a composite system formed by two irreps $\mathcal{Q}^d_\mu$ and $\mathcal{Q}^d_{(1)}$, decomposing $\mathcal{Q}^d_\mu \otimes \mathcal{Q}^d_{(1)}$ into irreps $\mathcal{Q}^d_\lambda$.
- Thus, given any direct sum of irreps

$$\bigoplus_\mu \mathcal{Q}^d_\mu \tag{4.3}$$

  we can decompose the tensor product of this sum with the defining irrep $\mathcal{Q}^d_{(1)}$ (representing the addition of a new qudit) into irreps by applying the CG transform of the appropriate dimension to term in the direct sum: that is, we can decompose

$$\left(\bigoplus_\mu \mathcal{Q}^d_\mu\right) \otimes \mathcal{Q}^d_{(1)} = \bigoplus_\mu \left(\mathcal{Q}^d_\mu \otimes \mathcal{Q}^d_{(1)}\right) \tag{4.4}$$

  into irreps by decomposing each term in the direct sum on the right-hand side independently, using the methods described in §3.2 and §3.3.
- But such a direct sum of irreps is exactly the output of the Schur transform, so given an implementation of the Schur transform on n qudits...

$$\left(\mathbb{C}^d\right)^{\otimes n} \overset{\mathcal{U}_d \times S_n}{\cong} \bigoplus_\mu \mathcal{P}_\mu \otimes \mathcal{Q}^d_\mu \tag{4.5}$$

  for $\mu \vdash n$, we can lift to a Schur transform on $n+1$ qudits by applying the CG transform to each irrep $\mathcal{Q}^d_\mu$ in the direct sum on the right-hand side, *taking the $\mathcal{P}_\mu$ simply as multiplicity spaces*. That is, we can write

$$\left(\mathbb{C}^d\right)^{\otimes n} \otimes \mathbb{C}^d \overset{\mathcal{U}_d \times S_n}{\cong} \left(\bigoplus_\mu \mathcal{P}_\mu \otimes \mathcal{Q}^d_\mu\right) \otimes \mathcal{Q}^d_{(1)} = \bigoplus_\mu \mathcal{P}_\mu \otimes \left(\mathcal{Q}^d_\mu \otimes \mathcal{Q}^d_{(1)}\right) \tag{4.6}$$

  where the first step represents the Schur transform on the first n qudits, and the second step simply shows how we can add the new $(n+1)$th qudit to each term separately, allowing us to apply the CG transform to each of those terms. By Schur duality, we know that decomposing the full tensor product into irreps of the unitary group will also decompose it into irreps of the symmetric group as multiplicity spaces, so we can ignore the irreps $\mathcal{P}_\mu$ entirely except as means for counting copies of the irreps $\mathcal{Q}^d_\mu$. This will be key to our analysis later.

4.1.1 Example

Let us continue the example presented at the end of the last chapter. Recall that when we decompose a composite system of two qubits into irreps using the CG transform, the results may be written down as in fig. 3.1 (§3.3). First note that this initial CG transform *is* the Schur transform on two qubits: any CG transform on two identical particles is the same as the Schur transform on those particles.

We wish to add a third qubit to the system and decompose the result into its irreps. So at the CG transform level, we have two combinations to concern ourselves with: $\mathcal{Q}^2_{(2)} \otimes \mathcal{Q}^2_{(1)}$ and $\mathcal{Q}^2_{(1,1)} \otimes \mathcal{Q}^2_{(1)}$, which to a physicist are $(\text{spin-}1) \otimes (\text{spin-}1/2)$ and $(\text{spin-}0) \otimes (\text{spin-}1/2)$, respectively. But we have already written down the CG transform for $(\text{spin-}1) \otimes (\text{spin-}1/2)$ (see (3.30), in §3.2), and a spin-0 particle is 1-dimensional, so that the CG transform on $(\text{spin-}0) \otimes (\text{spin-}1/2)$ is the 2×2 identity matrix. Combining these as a direct sum gives us S-CG$_2$, the *super-CG* transform that lifts our simple CG transform on two qubits (which, as we just noted, is the Schur transform on two qubits) to a Schur transform on three qubits:

$$\text{S-CG}_2 \quad = \quad \left[\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{\frac{1}{3}} & \sqrt{\frac{2}{3}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{\frac{2}{3}} & \sqrt{\frac{1}{3}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -\sqrt{\frac{2}{3}} & \sqrt{\frac{1}{3}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{\frac{1}{3}} & -\sqrt{\frac{2}{3}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \tag{4.7}$$

The CG transform on $\mathcal{Q}^2_{(2)} \otimes \mathcal{Q}^2_{(1)}$ is highlighted in red, and the CG transform on $\mathcal{Q}^2_{(1,1)} \otimes \mathcal{Q}^2_{(1)}$ is highlighted in blue. We order S-CG$_1$ (which is in fact just a simple CG transform) by the output irreps (i.e., triplet states first, followed by singlet state):

$$\text{S-CG}_1 \quad = \quad \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \end{array}\right] \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.8}$$

Multiplying the two gives us the Schur transform on three qubits:

$$\text{S-CG}_2 \cdot \text{S-CG}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & 0 & \sqrt{\frac{1}{3}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{\frac{1}{3}} & 0 & \sqrt{\frac{1}{3}} & \sqrt{\frac{1}{3}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & -\sqrt{\frac{2}{3}} & \sqrt{\frac{1}{6}} & 0 & \sqrt{\frac{1}{6}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{\frac{1}{6}} & 0 & \sqrt{\frac{1}{6}} & -\sqrt{\frac{2}{3}} & 0 \\ 0 & 0 & \sqrt{\frac{1}{2}} & 0 & -\sqrt{\frac{1}{2}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{\frac{1}{2}} & 0 & -\sqrt{\frac{1}{2}} & 0 & 0 \end{bmatrix} \tag{4.9}$$

where the input and output column vectors have the form:

$$\text{input:} \quad \left\{ \begin{array}{c} |\frac{1}{2},\frac{1}{2},\frac{1}{2}\rangle \\ |\frac{1}{2},\frac{1}{2},-\frac{1}{2}\rangle \\ |\frac{1}{2},-\frac{1}{2},\frac{1}{2}\rangle \\ |\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\rangle \\ |-\frac{1}{2},\frac{1}{2},\frac{1}{2}\rangle \\ |-\frac{1}{2},\frac{1}{2},-\frac{1}{2}\rangle \\ |-\frac{1}{2},-\frac{1}{2},\frac{1}{2}\rangle \\ |-\frac{1}{2},-\frac{1}{2},-\frac{1}{2}\rangle \end{array} \right\}, \qquad \text{output:} \quad \left\{ \begin{array}{c} |j=\frac{3}{2},m=\frac{3}{2},\mu=(2)\rangle \\ |j=\frac{3}{2},m=\frac{1}{2},\mu=(2)\rangle \\ |j=\frac{3}{2},m=-\frac{1}{2},\mu=(2)\rangle \\ |j=\frac{3}{2},m=-\frac{3}{2},\mu=(2)\rangle \\ |j=\frac{1}{2},m=\frac{1}{2},\mu=(2)\rangle \\ |j=\frac{1}{2},m=-\frac{1}{2},\mu=(2)\rangle \\ |j=\frac{1}{2},m=\frac{1}{2},\mu=(1,1)\rangle \\ |j=\frac{1}{2},m=-\frac{1}{2},\mu=(1,1)\rangle \end{array} \right\} \tag{4.10}$$

where in the output, μ indicates the irrep to which the first two qubits belonged. Alternatively, we can write the input and output vectors in a form that more clearly illustrates the mathematical structure of what is happening (for our inputs we now use 1 and 2 to denote the basis vectors for a

single qubit):

$$\text{input:} \quad \left\{ \begin{array}{l} |111\rangle \\ |112\rangle \\ |121\rangle \\ |122\rangle \\ |211\rangle \\ |212\rangle \\ |221\rangle \\ |222\rangle \end{array} \right\}, \qquad \text{output:} \quad \left\{ \begin{array}{l} \left|\begin{array}{|c|c|c|}\hline 1&2&3\\\hline\end{array}, \begin{array}{|c|c|c|}\hline 1&1&1\\\hline\end{array}\right\rangle \\ \left|\begin{array}{|c|c|c|}\hline 1&2&3\\\hline\end{array}, \begin{array}{|c|c|c|}\hline 1&1&2\\\hline\end{array}\right\rangle \\ \left|\begin{array}{|c|c|c|}\hline 1&2&3\\\hline\end{array}, \begin{array}{|c|c|c|}\hline 1&2&2\\\hline\end{array}\right\rangle \\ \left|\begin{array}{|c|c|c|}\hline 1&2&3\\\hline\end{array}, \begin{array}{|c|c|c|}\hline 2&2&2\\\hline\end{array}\right\rangle \\ \left|\begin{array}{|c|c|}\hline 1&2\\\hline 3\\\cline{1-1}\end{array}, \begin{array}{|c|c|}\hline 1&1\\\hline 2\\\cline{1-1}\end{array}\right\rangle \\ \left|\begin{array}{|c|c|}\hline 1&2\\\hline 3\\\cline{1-1}\end{array}, \begin{array}{|c|c|}\hline 1&2\\\hline 2\\\cline{1-1}\end{array}\right\rangle \\ \left|\begin{array}{|c|c|}\hline 1&3\\\hline 2\\\cline{1-1}\end{array}, \begin{array}{|c|c|}\hline 1&1\\\hline 2\\\cline{1-1}\end{array}\right\rangle \\ \left|\begin{array}{|c|c|}\hline 1&3\\\hline 2\\\cline{1-1}\end{array}, \begin{array}{|c|c|}\hline 1&2\\\hline 2\\\cline{1-1}\end{array}\right\rangle \end{array} \right\} \tag{4.11}$$

where the output vectors have the form $|T, t\rangle$ for T a standard λ-tableau that labels the vector within the S_n-irrep $\mathcal{P}_\lambda$, and t a semistandard λ-tableau with entries in $\{1, 2\}$ that labels the vector within the $\mathcal{U}_d$-irrep $\mathcal{Q}_\lambda^d$ (see §2.5 and §2.6). Note that λ takes on both possible values for $\lambda \vdash 3$ (that is, $(3, 0)$ and $(2, 1)$; $(1, 1, 1)$ is not possible because it has degree greater than 2). Also note that eliminating the box containing 3 in each standard tableau T gives the correct value for μ as the tableau associated to the first two qubits (in (4.10)).

4.2 Orderings and multiplicities

In the example we just studied, we had to be careful that the vectors output by S-CG$_1$ acting on the first two qubits (4.8) were correctly ordered for input to S-CG$_2$, which added the third qubit (4.7). Recall that for a linear operator acting on column vectors, the form of the input determines the order of the columns in the operator matrix, and the order of the rows determines the form of the output. So ensuring the correct form for the input to (4.7) came down to correctly ordering the rows in (4.8).

This is a specific instance of one of the general tasks in our implementation of the Schur transform, which we have ignored until now, aside from mentioning it in the introduction. We need to structure the output of each iteration of the super-CG transform so that it feeds into the next iteration correctly. Fortunately, the solution is elegant as well as straightforward.

- First, we build each super-CG transform according to the assumption that its input is a register $\mathcal{R}$ of qudits already ordered by irreps (i.e., grouped into the $\mathcal{Q}_\lambda^d$, according to the order we

are about to describe), with the new qubit simply added to the end of each state as a tensor product.

- Then for our recursion step to work, we need to order the output of each super-CG transform by the irreps of the new total number of qudits, so that they are ready to function as the new register $\mathcal{R}$ in the next iteration.
- This method has the additional advantage that the final output of our Schur transform will be ordered by the irreps of the final total number of qudits.
- However, this picture is not yet complete, because...

There is a further complication, which will lead us to the structuring that makes our method efficient. The complication is that most of the irreps have multiplicities greater than one, so how to order them sensibly is not immediately obvious. A naive and bad way to order them would be to put all copies of the irrep associated to the first partition (in, say, lexicographic order, which is ordering by length of first row, then second, and so forth) first, followed by all copies of the second irrep, followed by all copies of the third, and so on. For example, for $n = 4$ this ordering would look like:

BAD ORDERING:

$\begin{array}{|c|c|c|c|}\hline 1&2&3&4\\\hline\end{array}$: $\mathcal{Q}^d_{(4)}$

$\begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4\\\cline{1-1}\end{array}$: $\mathcal{Q}^d_{(3,1)}$

$\begin{array}{|c|c|c|}\hline 1&2&4\\\hline 3\\\cline{1-1}\end{array}$: $\mathcal{Q}^d_{(3,1)}$

$\begin{array}{|c|c|c|}\hline 1&3&4\\\hline 2\\\cline{1-1}\end{array}$: $\mathcal{Q}^d_{(3,1)}$

$\begin{array}{|c|c|}\hline 1&2\\\hline 3&4\\\hline\end{array}$: $\mathcal{Q}^d_{(2,2)}$

$\begin{array}{|c|c|}\hline 1&3\\\hline 2&4\\\hline\end{array}$: $\mathcal{Q}^d_{(2,2)}$

where the elements (standard tableaux) of the multiplicity space for each irrep are given first. While this ordering might seem natural, it would require a super-CG transform for the next step that acts on each of these irreps independently, including all copies. Thus, the last super-CG transform in the recursion will have to act on the entire space. Consequently, since the overall Schur transform will act on blocks given by this same structure, the last super-CG transform will be as inefficient as the Schur transform itself would be if we calculated it classically and implemented it directly on the quantum computer. So clearly we need a better option.

The ordering we will actually use is illustrated by the following example, again for $n = 4$:

GOOD ORDERING:

$$\left\{ \begin{array}{l} \ytableaushort{1234}: \quad \mathcal{Q}^d_{(4)} \\ \ytableaushort{123,4}: \quad \mathcal{Q}^d_{(3,1)} \\ \ytableaushort{12,34}: \quad \mathcal{Q}^d_{(2,2)} \end{array} \right\} \left\{ \begin{array}{l} \text{Blank slot w/ size: dim} \left(\mathcal{Q}^d_{(4)} \right) \\ \ytableaushort{124,3}: \quad \mathcal{Q}^d_{(3,1)} \\ \ytableaushort{13,24}: \quad \mathcal{Q}^d_{(2,2)} \end{array} \right\} \left\{ \begin{array}{l} \text{Blank slot w/ size: dim} \left(\mathcal{Q}^d_{(4)} \right) \\ \ytableaushort{134,2}: \quad \mathcal{Q}^d_{(3,1)} \\ \text{Blank slot w/ size: dim} \left(\mathcal{Q}^d_{(2,2)} \right) \end{array} \right\} \tag{4.12}$$

Here is what we do:

- Create an ordering of the *inequivalent* irreps $\mathcal{Q}^d_\lambda$: these are defined by λ (only their dimensions are dependent on d), so we choose to order them lexicographically.
- Have as many copies of this irrep-ordering as there are copies of the irrep with the highest multiplicity. In our example above, there are three copies of the irrep $\mathcal{Q}^d_{(3,1)}$, so we make three copies of the whole irrep-ordering: the copies of the irrep-ordering are enclosed in the large curly brackets.
- There will be unused "irrep slots," which are sets of states of the same dimension as one of the irreps that does not have maximal multiplicity: we will refer to these sets of states as *ghost-irreps*, and they can be ignored, except to note that they exist. For example, in the case above the last slot in the last irrep-ordering is a ghost of $\mathcal{Q}^d_{(2,2)}$, since there are three copies of the irrep-ordering but only two copies of $\mathcal{Q}^d_{(2,2)}$.

We now explain why this ordering is powerful, which is the central concept behind our algorithm. Recall that a super-CG transform is a direct sum of simple CG transforms, each of which adds a single qudit to some preexisting irrep $\mathcal{Q}^d_\mu$. For example, S-CG$_2$, given in (4.7), is made of a CG transform (the red block) that adds one qubit to the irrep $\mathcal{Q}^2_{(2)}$ and another CG transform (the blue block) that adds one qubit to the irrep $\mathcal{Q}^2_{(1,1)}$. Note that in this case there is one copy of each, but in the general case there are many copies of most of the irreps.

The key point is that when there are multiple copies of the same irrep, although the states they contain may be different, the CG transform that adds one qudit is the same for all of them. Therefore, we only need to implement each distinct CG transform once, as long we tensor them into an

identity matrix of sufficient dimension, and ensure that the input irreps feed into the appropriate "slots" in the resulting operator.

That is, when we write down the super-CG transform in block diagonal form, all the blocks corresponding to copies of the same irrep are the same. If we were to implement them all separately, we would end up implementing a unitary matrix with a number of off-diagonal elements that scales polynomially with the side-length of the matrix, i.e. exponentially with n, the number of qudits. That is what our bad ordering above would do, and this gives no advantage over performing the entire calculation classically and them decomposing the Schur transform directly as a single operation.

However, by using our ordering, we only need to implement the CG transform that adds one qudit to a particular irrep *once*. We build a direct sum of the CG transforms for each of the *inequivalent* irreps (i.e., for each element in one of our irrep-orderings, e.g. in (4.12)), and then copy that over all the irrep-orderings by taking tensor product with the identity matrix of appropriate dimension. For example, using the ordering given in (4.12), we build the following super-CG transform:

$$\text{S-CG}_{\text{example}} \begin{cases} CG\left(\mathcal{Q}^d_{(4)} \otimes \mathcal{Q}^d_{(1)}\right) \\ \quad \oplus \\ CG\left(\mathcal{Q}^d_{(3,1)} \otimes \mathcal{Q}^d_{(1)}\right) \\ \quad \oplus \\ CG\left(\mathcal{Q}^d_{(2,2)} \otimes \mathcal{Q}^d_{(1)}\right) \end{cases} \tag{4.13}$$

where $CG\left(\mathcal{Q}^d_{\mu} \otimes \mathcal{Q}^d_{(1)}\right)$ refers to the simple CG transform that adds a single qudit to the irrep $\mathcal{Q}^d_{\mu}$ (we have written the direct sum of these vertically to mimic the structure of the irrep-ordering in (4.12)). We then make three copies of this super-CG transform by taking the tensor product with a 3×3 identity matrix, so that the super-CG transform is applied to each copy of the irrep-ordering:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \text{S-CG}_{\text{example}} \tag{4.14}$$

Note that the super-CG transform acts on all the irrep slots in each copy of the ordering, including the ghost-irreps, but this is unimportant since we ignore them anyway.

4.2.1 Example

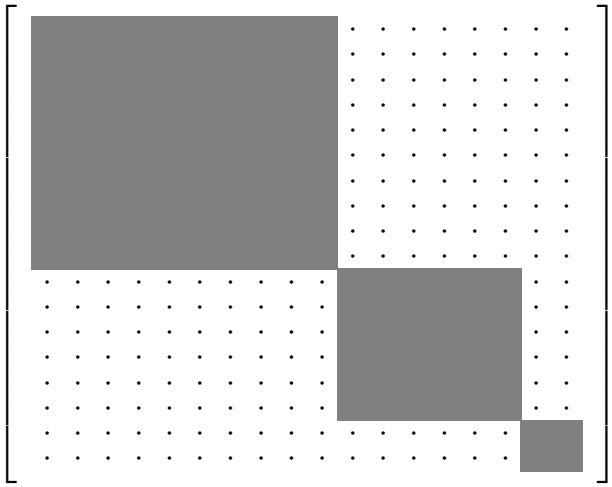
Let us continue to use the $n = 4$ case, and consider $d = 2$: this allows us to get definite dimensions for the irreps, which are...

$$\begin{aligned} \dim\left(\mathcal{Q}^d_{(4)}\right) &= 5 \\ \dim\left(\mathcal{Q}^d_{(3,1)}\right) &= 3 \\ \dim\left(\mathcal{Q}^d_{(2,2)}\right) &= 1 \end{aligned} \tag{4.15}$$

(Recall that these are calculated by counting the numbers of semistandard tableaux with entries in $\{1, 2\}$ (see §2.5)). We will not actually calculate the CG transforms, since we know how to calculate the transforms classically (§3.2 and §3.3). It is sufficient here to simply use the numbers given above: in the notation of (4.13),

$$\begin{aligned} &CG\left(\mathcal{Q}^d_{(4)} \otimes \mathcal{Q}^d_{(1)}\right) \text{ is } 10 \times 10, \\ &CG\left(\mathcal{Q}^d_{(3,1)} \otimes \mathcal{Q}^d_{(1)}\right) \text{ is } 6 \times 6, \text{ and} \\ &CG\left(\mathcal{Q}^d_{(2,2)} \otimes \mathcal{Q}^d_{(1)}\right) \text{ is } 2 \times 2 \end{aligned} \tag{4.16}$$

The side length of each of these matrices is double the dimension of the irrep it begins with, since we add an additional qubit to each. Thus the super-CG transform has the form



(4.17)

where the dots represent 0s, and all nonzero elements are inside the blocks. We take the tensor product of the 3×3 identity matrix with this to get our full operation, as in (4.14). The input is in the form (4.12), with dimensions two times those given in (4.15). Thus, we apply the appropriate CG transform to every copy of every irrep, while only actually implementing the CG transform once for each inequivalent irrep, so our efficiency is determined by the latter number.

4.3 Analysis for qubits

We now show that our algorithm for the Schur transform on n qubits ($d = 2$) can be decomposed into $O(n^3)$ two-level unitary operators (also known as Givens rotations: these are unitaries with only two nonzero off-diagonal elements), each of which can be approximated by a sequence of operators from the Clifford+T set (see appendix B), which can be implemented fault-tolerantly. We can break down the number of two-level unitaries required to construct our algorithm as follows:

- The Schur transform on n qubits requires $n - 1$ super-CG transforms: from S-CG$_1$ to the S-CG$_{n-1}$.
- The S-CG$_k$ is block diagonal, with each block corresponding to the addition of 1 qubit to each irrep of k qubits. In particular, for $\mu = (\mu_1, \mu_2) \vdash k$, the block associated to the addition of 1 qubit to $\mathcal{Q}^2_\mu$ has side length equal to twice the dimension of $\mathcal{Q}^2_\mu$. Note that since $d = 2$, μ cannot have degree greater than 2, so we can write $\mu = (\mu_1, \mu_2)$ as long as we remember that μ_2 may be 0.
- The dimension of $\mathcal{Q}^2_\mu$ is given by the number of semistandard μ-tableaux t with entries in $\{1, 2\}$ (see theorem 2.5.1). This number is given by

$$\dim\left(\mathcal{Q}^2_\mu\right) = \mu_1 - \mu_2 + 1 \tag{4.18}$$

  We can see this by noting that the entire second row of t must be occupied by 2s, while the entries in the first row that are above 2s must be ones, since the columns must be strictly increasing downward. Thus t is defined by the position of the first 2 in the first row, which may be at indices $\mu_2 + 1$ through μ_1, giving $\mu_1 - \mu_2$ possible locations, or may not appear at all, giving one additional possible tableau.
- We can reduce a nonsingular square matrix to upper-triangular using one Givens rotation per nonzero entry below the main diagonal. If we reduce a unitary matrix to upper-triangular, we must have reduced it to a diagonal matrix whose diagonal entries have norm 1, since this is the form of any upper-triangular unitary matrix. By then applying a one-level phase shift to each main diagonal entry, we can map that diagonal matrix to the identity matrix, and since one-level rotations are a special case of two-level rotations, we can simply count these as Givens rotations as well. We refer to the total number of Givens rotations into which the goal matrix is decomposed as the *sequence length*.
- The number of nonzero entries on or below the main diagonal thus gives us our two-level gate sequence length for a single CG transform. This sum is bounded by the total number of nonzero elements in the CG transform (and is of the same order). Each row in the CG transform is an eigenvector of the total spin projection: suppose a particular row $\langle\phi|$ has total spin projection m. As discussed in §3.2.2, this means that for any computational basis vector $|m_1, m_2\rangle$ appearing in the linear combination that forms $|\phi\rangle$, $m_1 + m_2 = m$. But since our CG transforms always just add a single qubit to some irrep $\mathcal{Q}^2_\mu$, m_2 is restricted to the

values $+\frac{1}{2}$ or $-\frac{1}{2}$. Therefore, there are at most two computational basis vectors appearing in the linear combination that forms $|\phi\rangle$, i.e., there are at most two nonzero entries per row in the CG transform. Therefore, the two-level gate sequence length for the CG transform is bounded by 2ℓ, where ℓ is the side length of the CG transform.

- In our case, $\ell = 2(\mu_1 - \mu_2 + 1)$, so the sequence length for the CG transform on $\mathcal{Q}^2_\mu$ is bounded by
$$4(\mu_1 - \mu_2 + 1) = 4(k - 2\mu_2 + 1) \tag{4.19}$$
for $\mu_1 + \mu_2 = k$.

- S-CG$_k$ is a direct sum of CG transforms on $\mathcal{Q}^2_\mu$, for every $\mu \vdash k$. Since each μ has degree 2, the possibilities are
$$\mu \in \left\{ (k - \mu_2, \mu_2) \mid \mu_2 \in \left\{ 0, 1, 2, ..., \left\lfloor \frac{k}{2} \right\rfloor \right\} \right\} \tag{4.20}$$

- Therefore, the sequence length for S-CG$_k$ is
$$4 \sum_{\mu_2=0}^{\lfloor k/2 \rfloor} (k - 2\mu_2 + 1) \tag{4.21}$$

- Our Schur transform decomposes into S-CG$_k$ for $k = 1, 2, ..., n-1$, as discussed above, so the two-level gate sequence length for the full Schur transform is
$$\boxed{4 \sum_{k=1}^{n-1} \sum_{\mu_2=0}^{\lfloor k/2 \rfloor} (k - 2\mu_2 + 1)} \tag{4.22}$$
which is $O(n^3)$, as desired.

In practice, there is another "rearranger" step involved in implementing the Schur transform, but as we argue in §4.4.2, its contribution to the two-level gate sequence length is also $O(n^3)$. In appendix C, we give Mathematica code that implements the Schur transform compiler described in pseudocode in the next section. Using this code, we can evaluate the actual sequence lengths and compare them to the theoretical bounds.

In fig. 4.1, the black line is n^3, and the points are the two-level gate sequence lengths generated by our implementation of the Schur transform for qubits.

Now that we have our sequence of $O(n^3)$ two-level rotations, we want to approximate each two-level rotation by Clifford+T operators. Using known methods ([36] or [37], for example; see appendix B for more detail), a two-level rotation on n qubits can be decomposed into $O(n \log(1/\delta))$ Givens rotations, with error δ. If we want to achieve an overall error bounded by ϵ, we can calculate the error bound required for the individual two-level rotations as follows:

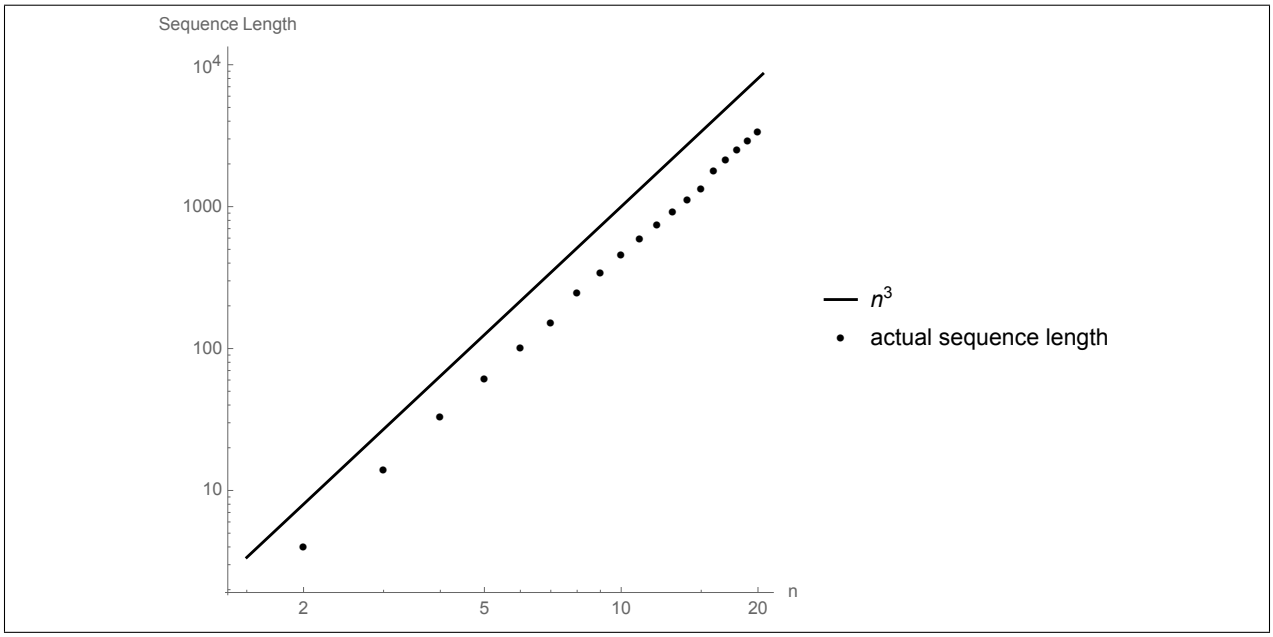


Figure 4.1: Two-level gate sequence lengths for qubits

Lemma 4.3.1 :
Let $A_1, A_2, ..., A_m$ be a sequence of unitary operators. If $B_1, B_2, ..., B_m$ is a sequence of unitary operators such that B_j approximates A_j with error δ in the trace norm for all j, then the product $\Pi_{j=1}^{m} B_j$ approximates the product $\Pi_{j=1}^{m} A_j$ with error $\epsilon \leq m\delta$.

Proof. See appendix A. □

Therefore, if we want our overall error to be bounded by ϵ, the errors for our decomposition matrices (the two-level rotations) must be bounded by $\delta = \epsilon/m$. We decompose into $m = O(n^3)$ two-level rotations, so our decomposition errors for the individual rotations must be bounded by $\frac{\epsilon}{an^3}$ for some scalar a. Thus the lengths of the approximating sequences will be bounded by

$$O\left(n \log\left(\frac{an^3}{\epsilon}\right)\right) = O\left(3n \log\left(\frac{n}{\epsilon}\right)\right) = O\left(n \log\left(\frac{n}{\epsilon}\right)\right) \tag{4.23}$$

Multiplying by the number of two-level gates in our decomposition gives us the overall sequence length for decomposition of the Schur transform on n qubits:

$$O\left(n^4 \log\left(\frac{n}{\epsilon}\right)\right) \tag{4.24}$$

4.4 Practical implementation

In this section we give complete explanations and pseudocode for an implementation of the Schur transform on n qubits according to the abstract plan laid out above. In appendix C, we give Mathematica code for this Schur transform compiler. It is a compiler in the sense that it is a classical program that returns a sequence of Clifford+T gates, which will implement the Schur transform if executed on a quantum computer. In the pseudocode in this section, comments are given in red text, and are preceded by #.

4.4.1 Clebsch-Gordan transform

Below is pseudocode to calculate the CG transform, using the "physicist's perspective" implementation described in §3.2. Given a partition μ, the following algorithm returns the matrix for the CG transform on $\mathcal{Q}^2_\mu \otimes \mathcal{Q}^2_{(1)}$. Since μ has degree at most 2, assume that μ has degree exactly 2 (i.e., input $(\mu_1, 0)$ if $\mu_2 = 0$, rather than (μ_1)).

1. **CG**(μ):
2. $j = \frac{\mu_1 - \mu_2}{2}$ # j is the effective total spin for $\mathcal{Q}^2_\mu$ (since $2j + 1 = \mu_1 - \mu_2 + 1$)
3. # Get first state with total spin $j + \frac{1}{2}$:
4. $|j + \frac{1}{2}, j + \frac{1}{2}\rangle = |j\rangle \otimes |\frac{1}{2}\rangle$
5. # Get rest of states with total spin $j + \frac{1}{2}$:
6. $m = j - \frac{1}{2}$
7. **while** $m \geq -j - \frac{1}{2}$:
8. $|j + \frac{1}{2}, m\rangle = \hat{a}_- |j + \frac{1}{2}, m + 1\rangle$ # $\hat{a}_-$ is the lowering operator
9. $m = m - 1$
10. # Get first state with total spin $j - \frac{1}{2}$:
11. $a = (\langle j| \otimes \langle -\frac{1}{2}|)|j + \frac{1}{2}, j - \frac{1}{2}\rangle$ # a is the coefficient of $|j\rangle \otimes |-\frac{1}{2}\rangle$ in $|j + \frac{1}{2}, j - \frac{1}{2}\rangle$
12. $b = (\langle j - 1| \otimes \langle \frac{1}{2}|)|j + \frac{1}{2}, j - \frac{1}{2}\rangle$ # b is the coefficient of $|j-1\rangle \otimes |\frac{1}{2}\rangle$ in $|j + \frac{1}{2}, j - \frac{1}{2}\rangle$
13. $|j - \frac{1}{2}, j - \frac{1}{2}\rangle = b|j\rangle \otimes |-\frac{1}{2}\rangle - a|j - 1\rangle \otimes |\frac{1}{2}\rangle$ # the state with $m = j - \frac{1}{2}$ that is orthogonal to $|j + \frac{1}{2}, j - \frac{1}{2}\rangle$
14. # Get rest of states with total spin $j - \frac{1}{2}$ (if any):
15. **if** $j \neq \frac{1}{2}$:
16. $m = j - \frac{3}{2}$

17. **while** $m \geq -j + \frac{1}{2}$:
18. $|j - \frac{1}{2}, m\rangle = \hat{a}_- |j - \frac{1}{2}, m + 1\rangle$
19. $m = m - 1$
20. # List of input states in correct ordering:
21. $in = \left(\langle j| \otimes \langle \frac{1}{2}|,\ \langle j| \otimes \langle -\frac{1}{2}|,\ \langle j-1| \otimes \langle \frac{1}{2}|,\ \langle j-1| \otimes \langle -\frac{1}{2}|,\ \langle j-2| \otimes \langle \frac{1}{2}|, ...\right)$
22. # List of output states in correct ordering:
23. $out = \left(|j + \frac{1}{2}, j + \frac{1}{2}\rangle, |j + \frac{1}{2}, j - \frac{1}{2}\rangle, ..., |j + \frac{1}{2}, -j - \frac{1}{2}\rangle, |j - \frac{1}{2}, j - \frac{1}{2}\rangle, ..., |j - \frac{1}{2}, -j + \frac{1}{2}\rangle\right)$
24. # Output matrix (denoted by entries):
25. $mat[[i, j]] = in[[j]]out[[i]]$
26. **return** mat

4.4.2 Schur Transform

In §4.2, we described how to order the irreps (including some ghost irreps that act as placeholders) in order to make our CG transform (the iteration step in the Schur transform implementation) as efficient as possible. A example of this ordering is given in (4.12). We have an implementation of the CG transform; our main remaining task is to describe how the general ordering in §4.2 can be efficiently encoded in a register of qubits, and how the CG transform is to be applied correctly to this encoding.

For our encoding, we divide the register of qubits into three main subregisters:

- a subregister called ***seq*** (for sequence: recall §2.6 (equation (2.42))) that encodes the multiplicities of the irreps.
- a subregister called ***par*** (for partition) that encodes the identities of the inequivalent irreps.
- a subregister called ***stat*** (for states) that encodes the bases of the irreps.

Thus, the example ordering in (4.12) shows the sets to be encoded in the **seq** and **par** qubits: that is, it shows the irreps and their multiplicities, but not the individual states within them. To encode these sets in qubits, we begin with a simple idea: for a set containing k objects, we can encode the objects in the set using the first k states of $\lceil \log_2(k) \rceil$ qubits.

We need to complicated this slightly: we want to use the same subregister, **stat**, to encode the basis of every irrep, and the same subregister, **seq**, to encode the multiplicities of every irrep. Therefore, the **stat** qubits need to be sufficient to encode the basis of the irrep with largest dimension, the **seq** qubits need to be sufficient to encode the largest multiplicity of any irrep. The **par** qubits have no added complication: they just need to be sufficient to encode the identities of the irreps. Thus, if d

is the highest dimension of any irrep, m is the highest multiplicity of any irrep, and q is the number of inequivalent irreps, the sizes of the subregisters (numbers of qubits) are

$$|\mathbf{seq}| \geq \lceil \log_2(m) \rceil \tag{4.25}$$
$$|\mathbf{par}| = \lceil \log_2(q) \rceil \tag{4.26}$$
$$|\mathbf{stat}| = \lceil \log_2(d) \rceil \tag{4.27}$$

We wrote $\geq$ for the number of **seq** qubits, because it will end up being simpler for us to use a larger number of the same order. We are now in a position to calculate these numbers:

When we have performed the Schur transform on n qubits, the state space should be decomposed into irreps as (2.36): we refer to this as having "symmetrizing" the qubits. As we have discussed, the irreps are labeled by partitions of n with degree less than or equal to d, which is 2 in the qubit case. The number of degree-2 partitions λ of n (counting the partition $(n, 0)$) is

$$q = \left\lfloor \frac{n}{2} \right\rfloor + 1 \tag{4.28}$$

which we can see by noting that the length of the second row, λ_2, can take on any value from 0 to $\lfloor n/2 \rfloor$. Therefore,

$$|\mathbf{par}| = \lceil \log_2 \left(\left\lfloor \frac{n}{2} \right\rfloor + 1 \right) \rceil \tag{4.29}$$

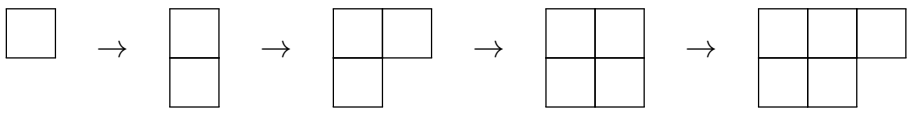
The dimension of $\mathcal{Q}_\lambda^2$ is the number of semistandard λ-tableaux with entries in $\{1, 2\}$, which, as we argued in §4.3, is given by $\lambda_1 - \lambda_2 + 1$. We want the maximum value of this function for any $\lambda \vdash n$, which is acheived when $\lambda = (n, 0)$, giving

$$d = n + 1 \tag{4.30}$$

Therefore,

$$|\mathbf{stat}| = \lceil \log_2(n + 1) \rceil \tag{4.31}$$

Lastly, the multiplicity of $\mathcal{Q}_\lambda^2$ is the number of standard λ-tableaux. This number can be calculated directly (one would use an expression called the *hook formula* (5.1)), but we instead make use of the following simplification. We can think of a standard λ-tableau as encoding an order of adding boxes in a Young diagram to build up to λ (recall §2.6). For example, under this interpretation, the standard tableau $\begin{array}{|c|c|c|}\hline 1 & 3 & 5 \\ \hline 2 & 4 \\ \cline{1-2}\end{array}$ corresponds to the following order of adding boxes:



(4.32)

Using this interpretation, note that each time we add a box we have at most 2 options for where to put it. When adding the first and last box we have no choice, so in adding n boxes to build a

particular Young diagram, we get to make at most $n-2$ binary choices in total. Thus, the maximum multiplicity of any irrep labeled by a partition of n is bounded by 2^{n-2}, so we will take that as our number of states to encode the multiplicities. Therefore, taking

$$|\mathbf{seq}| = n - 2 \tag{4.33}$$

will always be sufficient. Furthermore, it is not an overuse of ancillary qubits to apply this simplification, as we now show:

We know we need at least n encoding qubits in our whole register to encode the irreps of $(\mathcal{C}^2)^{\otimes n}$. If we add up the required qubits for our whole register (the sum of equations (4.29), (4.31), and (4.33)), we get

$$n_{\text{encoding}} = n + \lceil \log_2(n+1) \rceil + \lceil \log_2 \left(\left\lfloor \frac{n}{2} \right\rfloor + 1 \right) \rceil - 2 \tag{4.34}$$

$n = 2$ is our first case of interest, and for this case, $\lceil \log_2(n+1) \rceil = 2$ and $\lceil \log_2 \left(\lfloor \frac{n}{2} \rfloor + 1 \right) \rceil = 1$, so $n_{\text{encoding}} = n + 1$. Thus in general we have

$$n_{\text{encoding}} = n + f(n) \tag{4.35}$$

where $f(n) = O(\log_2(n))$ and $f(n) \geq 1$. Therefore, the number of qubits used for the encoding is optimal in the sense that the dominant order in n is linear and has a definite coefficient of 1, and the lower order terms are exponentially smaller and thus negligible for large n.

These numbers give the final sizes of **seq**, **par**, and **stat**, which will include all of the encoding qubits. However, not every super-CG transform in the recursion will use all of the encoding qubits, so the effective sizes **seq**, **par**, and **stat** are allowed to change during the algorithm as long as they reach their final values ((4.29), (4.31), and (4.33)) at the final iteration.

What about the recursion step itself? As noted in §4.2, we want the output states of each iteration to be ordered by the irreps of the total number of qubits. The input states should be in the form of the output of the previous step, with one new computational qubit tensored into the bottom of the register (this is the new qubit that will end up in **seq**). So the steps in the iteration are:

1. Tensor in a new qubit to the bottom of the register (containing $n-1$ qubits already), which we may assume is symmetrized.
2. Perform CG transforms on each irrep $\mathcal{Q}^2_\mu$ for $\mu \vdash (n-1)$. Recall that the outputs of the CG transforms as we have constructed them are ordered by the irreps $\mathcal{Q}^2_\lambda$ for $\lambda \vdash n$.
3. Rearrange the irreps $\mathcal{Q}^2_\lambda$ according to the ordering described in §4.2, now for n.

It is high time for an example: let us consider the iteration $n = 4 \rightarrow n = 5$. We first show the forms of the state vectors at each step of the transform, and then show the actual transform matrices:

Example: vector form

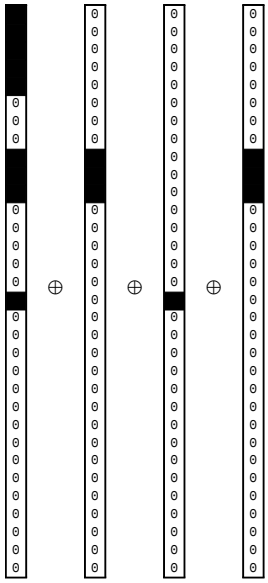
By (4.29), (4.31), and (4.33), initially we have

$$|\mathbf{seq}| = 2, \qquad |\mathbf{par}| = 2, \qquad |\mathbf{stat}| = 3 \tag{4.36}$$

Note: $|\mathbf{par}|$ and $|\mathbf{stat}|$ may be larger, since in the context of the larger Schur transform they will be set to their final values, as discussed above, but they are at least as large as (4.36). The irreps for $n = 4$ are

$$\begin{aligned} \mathcal{Q}^2_{(4,0)} : &\quad m = 1, \quad d = 5 \\ \mathcal{Q}^2_{(3,1)} : &\quad m = 3, \quad d = 3 \\ \mathcal{Q}^2_{(2,2)} : &\quad m = 2, \quad d = 1 \end{aligned} \tag{4.37}$$

Thus our input vector might look like



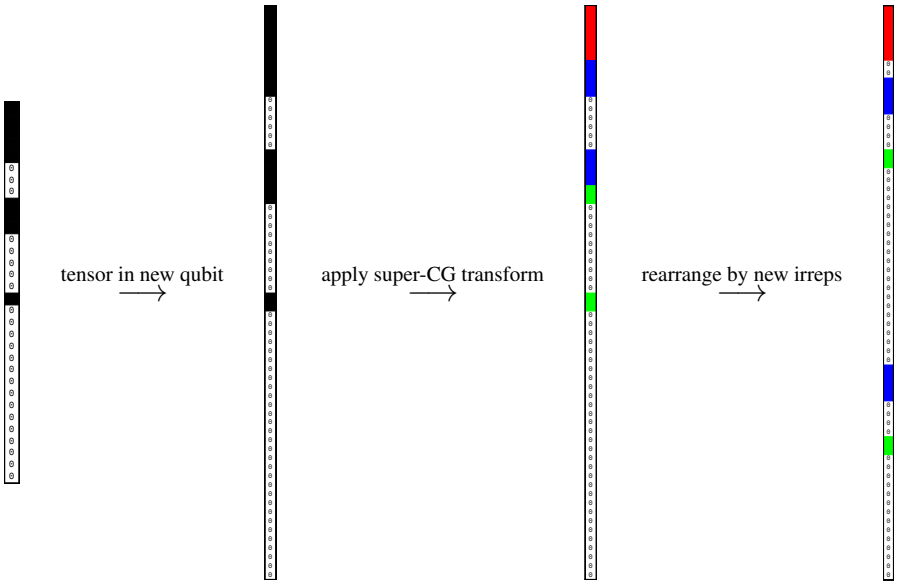
(4.38)

Each column here represents one state of **seq**. Each column is divided into quarters representing the states of **par**. Thus each quarter of a column encodes an inequivalent irrep: there is one "slot" for each inequivalent irrep in each column. The locations of the actual encoded states are highlighted by the black squares. Which slots are filled is determined by the outputs of previous super-CG iterations, but note that each irrep is in the correct slot within its column (for example, the 3-dimensional irrep, $\mathcal{Q}^2_{(3,1)}$, always appears in the second encoding slot within a column).

We tensor in the new qubit to the bottom of the register, then apply the appropriate CG transform to each irrep, copied over the values of **seq**: thus in this case the super-CG step will be

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \otimes \text{super-CG} \tag{4.39}$$

We then rearrange the output by the new irreps (for $n = 5$). Below is a schematic for this whole process, using the first column in (4.38) as our example, since the same operations happen for each column, until the final reordering:



(4.40)

where in the second to last and last columns the colors represent the new irreps (for $n = 5$): these are

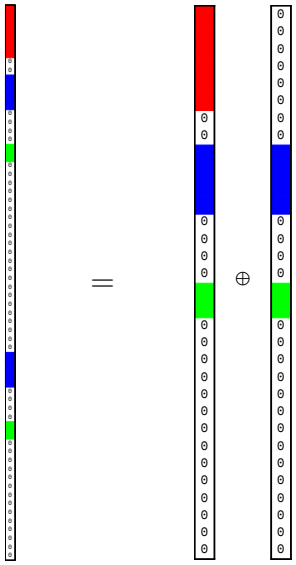
$$\begin{aligned} \mathcal{Q}^2_{(5,0)}: &\quad m = 1, \quad d = 6, \quad \text{red in (4.40)} \\ \mathcal{Q}^2_{(4,1)}: &\quad m = 4, \quad d = 4, \quad \text{blue in (4.40)} \\ \mathcal{Q}^2_{(3,2)}: &\quad m = 5, \quad d = 2, \quad \text{green in (4.40)} \end{aligned} \tag{4.41}$$

The schematic (4.40) is our clearest picture thus far of the structure of our algorithm, so it is worth reiterating how the process fits together. First, the process in (4.40) represents the iteration step that is applied to *each* state of **seq**, i.e., to each column in (4.38). Each such iteration step adds one qubit to the register, but the exponential growth in the dimension of the register is absorbed into the number of columns (when written in the form of (4.38)) in the input to each iteration; so the height of the individual columns, which determines the complexity of the iteration step, is subexponential.

This is why our algorithm is efficient!

We can see that the exponential growth in the dimension is absorbed into **seq** but noting that, although the output column in (4.40) has twice the dimension of the input column, it also contains two copies of each inequivalent irrep slot, so that in fact the new qubit has become part of **seq**. To

make this completely explicit:



(4.42)

Thus when we have performed the iteration step (4.40) on each column in the input (4.38), if we rewrite the output in the form (4.38), we will have 8 columns instead of 4, but their heights will be the same. The heights of the columns (determined by the number of qubits in **par** and **stat**) only change "logarithmically often," as given in (4.29) and (4.31).

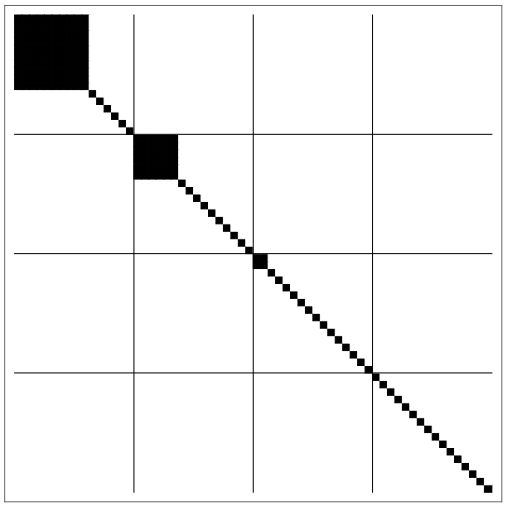
Example: matrix form

We can also write down these transformations in terms of matrices. We assume, as above, that input is in the form

$$(\text{output of Schur transform on } 4 \text{ qubits}) \otimes (\text{one new qubit}) \tag{4.43}$$

This input is already in the correct form for input to the super-CG transform: this preserves states within their irreps, so the whole operation has block diagonal form given by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \otimes$$



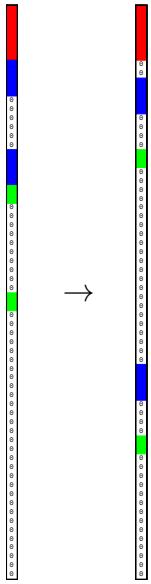
(4.44)

where the black squares in the super-CG operator (the second array) represent the possible locations of nonzero entries in the matrix, and the dividing lines in the super-CG operator demarcate the three actual irreps and one ghost irrep.

Next, we need to rearrange the states according to the irreps for $n = 5$. The new subregister sizes (from (4.29), (4.31), and (4.33)) are

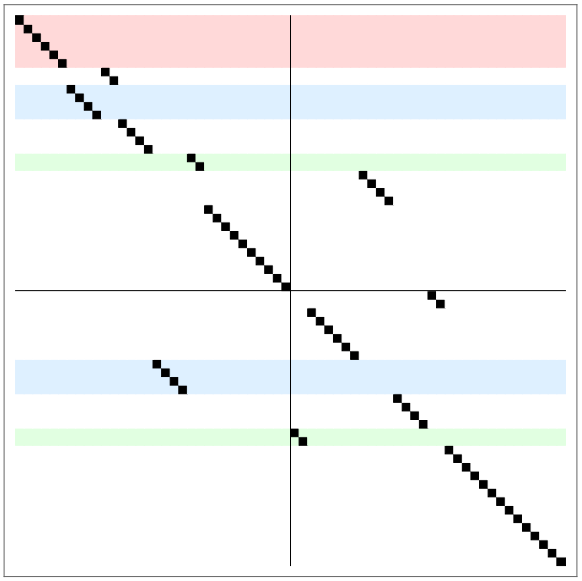
$$|\mathbf{seq}| = 3, \qquad |\mathbf{par}| = 2, \qquad |\mathbf{stat}| = 3 \tag{4.45}$$

Currently, (not showing multiplicities) our state vector has the form of the second to last vector in (4.40), so we need to perform the final permutation:



(4.46)

The matrix that performs this permutation is



(4.47)

where the black boxes represent nonzero entries (in this case, all 1s), and the pale highlighting indicates the rows that contain the output irreps, using the same color scheme as in (4.41). The states of **par** are again separated by the fine dividing lines.

As we noted above, although the input to (4.47) (the first vector in (4.46)) only represents one state of **seq** (one copy of each irrep), the output (the second vector in (4.46)) contains two states (two copies of each irrep, counting ghost irreps). This is thus the step where the new qubit is moved to **seq**: the output vector of (4.47) (the second vector in (4.46)) becomes a *pair* of columns when the output is written in the same form as (4.38).

At this point the reader might be concerned, because this rearranging step was not included in our analysis above (§4.3). The reasons for that are, first, that justifying and explaining the rearranging step is complex (see above), and second, that its contribution to the overall sequence length turns out have the same bound as the super-CG transforms, as we now argue:

1. First, note that the rearranging permutation matrix, like the super-CG transform itself, can be tensored over all the **seq** states (of the initial **seq**).

2. The rearranging permutation only needs to move the states that encode the irreps, plus the states at the locations they need to go to if those are not otherwise being moved, so the total number of states to be moved is at most

$$2(\text{sum of dimensions of irreps}) \tag{4.48}$$

3. The number of off-diagonal elements in the super-CG transform also scales like the sum of the dimensions of the irreps, so the sequence length (4.48) is asymptotically identical to that associated to the super-CG transform.

Once the rearrange step has been performed, the output vectors have the correct ordering (as discussed above: see §4.2 and (4.12)) for the total number of qubits, so we are done. As discussed in §4.3, our algorithm decomposes the Schur transform on n qubits into a sequence of $O(n^3)$ Givens rotations (the exact number is given by (4.22)). Each of these can be decomposed using the methods discussed in appendix B into $O(n \log(n/\epsilon))$ Clifford+T gates, giving a total sequence length of

$$O\left(n^4 \log\left(\frac{n}{\epsilon}\right)\right) \tag{4.49}$$

for an overall accuracy of ϵ.

Pseudocode

In this section we give pseudocode for our algorithm to compile the Schur transform on n qubits into a product of Clifford+T gates. First we need a subroutine to compute S-CG$_n$. We call it $CGBlocks$, since it returns a block-diagonal matrix whose blocks are simple CG transforms. $nstat$ should take the number of qubits being used to encode **stat**. If this input is smaller than the required minimum number (4.31), then the output matrix will be invalid.

1. **CGBlocks**$(n, nstat)$:

2. $npar = nstat - 1$
3. $pars =$ list of degree 2 partitions of n, in top-down lexicographic order
4. $dstat =$ list of numbers of semi-standard tableaux with entries in $\{0, 1\}$ associated to each partition in $pars$
5. $mat =$ identity matrix of dimension $2^{nseq+npar+nstat}$
6. # Iterate over irreps:
7. **for** i **from** 0 **to** $length(pars) - 1$:
8. # Upper left corner location for current block:
9. $start = j(2^{npar+nstat+1}) + i(2^{nstat+1})$
10. # Insert simple CG transform:
11. $cg = CG(dstat[[i]])$
12. **for** k, l **from** 0 **to** $2dstat[[i]] - 1$:
13. $mat[[start + k, start + l]] = cg[[k, l]]$
14. **return** mat

Next, we need a subroutine to reorder the output vectors from the super-CG transform according to the scheme described above (§4.2). The output matrix will be associated to the addition of one new qubit to n already-symmetrized qubits. $nstat$ should take the number of qubits being used to encode **stat**. If this input is smaller than the required minimum number (4.31), then the output matrix will be invalid.

1. **CGRearrange**$(n, nstat)$:
2. $npar = nstat - 1$
3. $pars =$ list of degree 2 partitions of n, in top-down lexicographic order
4. $dstat =$ list of numbers of semi-standard tableaux with entries in $\{0, 1\}$ associated to each partition in $pars$
5. $line1 = \{\}$ # will contain the first line of the permutation to be implemented (in two-line notation)
6. $line2 = \{\}$ # will contain the second line of the permutation to be implemented (in two-line notation)
7. $m =$ list of 0s: one for each element of pars, plus one extra # keeps track of the numbers of each output irrep that have been moved to their correct locations so far.

8. # Iterate over irreps:
9. **for** i **from** 0 **to** $length(pars) - 1$:
10. # Move first output irrep appearing in ith input irrep (lines 13-22):
11. # Location of upper left corner of block (x coordinate):
12. $xstart = j(2^{npar+nstat+1}) + i(2^{nstat+1})$
13. # Location of upper left corner of block (y coordinate):
14. $ystart = m[[i]](2^{npar+nstat}) + i(2^{nstat})$
15. # Update m: just moved one copy of the ith output irrep.
16. $m[[i]] = m[[i]] + 1$
17. # Add permutations to lists: this output irrep has dimension $dstat[[i]] + 1$
18. **for** k **from** 0 **to** $dstat[[i]]$:
19. **push** $xstart + k$ **to** $line1$
20. **push** $ystart + k$ **to** $line2$
21. # Move second output irrep appearing in ith input irrep (lines 24-33):
22. # Location of upper left corner of block (x coordinate):
23. $xstart = j(2^{npar+nstat+1}) + i(2^{nstat+1})$
24. # Location of upper left corner of block (y coordinate):
25. $ystart = m[[i+1]](2^{npar+nstat}) + (i+1)(2^{nstat})$
26. # Update m: just moved one copy of the $(i+1)$th output irrep.
27. $m[[i+1]] = m[[i+1]] + 1$
28. # Add permutations to lists: this output irrep has dimension $dstat[[i]] - 1$
29. **for** k **from** 0 **to** $dstat[[i]] - 2$:
30. **push** $xstart + k$ **to** $line1$
31. **push** $ystart + k$ **to** $line2$
32. # Permute unmatched indices in $line2$ to unmatched indices in $line1$:
33. **for** i **from** 0 **to** $length(line1 \setminus line2) - 1$:
34. **push** $(line2 \setminus line1)[[i]]$ **to** $line1$
35. **push** $(line2 \setminus line2)[[i]]$ **to** $line2$

36. **return** the permutation matrix associated to the permutation given by $\binom{line1}{line2}$ (in two-line notation - indices that appear in neither line are not permuted).

Now we are ready, at last, to implement the Schur transform on n qubits, or more accurately, to compute the string of primitives that we should implement on our quantum computer to perform the Schur transform. The actual work has been done in the above two subroutines; all that remains is to put the pieces together. The following algorithm returns a list whose elements have the following form:

$$\{a, m, b\} \tag{4.50}$$

for a matrix m and two numbers a and b. To obtain the Schur transform, approximate each m according to the decomposition into Clifford+T operators given in appendix B, and then take the product over all the elements in this list of

$$\mathbf{I}_{2^a} \otimes (\text{approximation of } m) \otimes \mathbf{I}_{2^b} \tag{4.51}$$

where $\mathbf{I}_{2^x}$ denotes the identity matrix of dimension 2^x. The resulting Schur transform has the following structure: the input vector should be arranged with the computational qubits at the bottom of the register (the end of the tensor product), and the output vector will be in the symmetrized ordering (see above).

1. **Schur**(n):
2. $\quad nstat = \lfloor \log_2(n) \rfloor + 1$
3. $\quad$ # List **CGBlocks** and **CGRearrange** matrices from largest to smallest:
4. $\quad out = \{\}$
5. $\quad$ **for** i **from** $n-1$ **to** 1 **by** -1:
6. $\quad\quad$ **push** $\{i-1,$ **CGRearrange**$(i, nstat),\ n-1-i\}$ **to** out
7. $\quad\quad$ **push** $\{i-1,$ **CGBlocks**$(i, nstat),\ n-1-i\}$ **to** out
8. $\quad$ **return** out

To summarize, **Schur**(n) returns a list of matrices that, when tensored into the appropriate identity matrices and multiplied in the order given in the list, give the Schur transform on n qubits exactly. By decomposing the matrices in the list into primitives in the Clifford+T set, we obtain an approximation to the Schur transform with accuracy ϵ, in a total of

$$O\left(n^4 \log(n/\epsilon)\right) \tag{4.52}$$

primitive gates. If we denote our algorithm to decompose a unitary matrix into a product of Clifford+T operators by **ToCliffordT**(U, ϵ), which returns a list of Clifford+T operators to be multiplied, then we can write the full decomposition algorithm that takes a number of qubits n and returns a sequence of Clifford+T operators that approximate the Schur transform as

1. **SchurToCliffordT**(n, ϵ):
2. $cgs =$ **Schur**(n)
3. $out = \{\}$
4. **for** i **from** 0 **to** $Length(cgs)$:
5. $ctlist =$ **ToCliffordT**$(cgs[[i, 1]], \epsilon)$
6. **for** j **from** 0 **to** $Length(ctlist)$:
7. **push** $(\mathbf{I}_{2^{cgs[[i,0]]}} \otimes\ ctlist[[j]]\ \otimes \mathbf{I}_{2^{cgs[[i,2]]}})$ **to** out
8. **return** out

where $\mathbf{I}_{2^x}$ denotes the identity matrix of dimension 2^x.

Chapter 5

Conclusion

We have described a means of compiling the Schur transform on n qubits into a product of $O(n^3)$ Givens rotations, each of which can itself be decomposed into $O(n \log(n/\epsilon))$ Clifford+T gates, using the procedure described in appendix B, for an overall accuracy of ϵ. As discussed in appendix B, decomposing a general unitary operator into a product of primitives from any universal gate set (in this case the Clifford+T set) requires a sequence length that is exponential in n. Thus our implementation of the Schur transform is efficient, in the sense that the sequence length of Clifford+T gates is polynomial, not exponential.

We should also note that directly decomposing the Schur transform into Givens rotations does *not* give a polynomial sequence length: the resulting sequence length is exponential in n, though the Schur transform matrix is still sparse (the sequence length scales like the side length of the Schur transform matrix, not the area). To see this, it is sufficient to recall that the Schur transform contains one block for each *copy* of each irrep of n qubits, and the multiplicity of an irrep $\mathcal{Q}_\mu^2$ is given by the number of standard tableaux with shape μ. The *hook formula* gives the number of standard tableaux with a given shape [33]: for $\mu \vdash n$,

$$\text{number of standard tableaux} = \frac{n!}{\prod_{\text{all boxes}} \text{hook length of box}} \tag{5.1}$$

where the *hook length* of a box is the number of boxes to its right plus the number of boxes below it, counting itself once. In our case, μ has degree at most 2: let us consider the irrep labeled by $\mu = (n/2, n/2)$ for even n. For this case, (5.1) is given by

$$\text{number of standard tableaux} = \frac{n!}{\prod_{i=1}^{n/2}(i+1)i} = \frac{1}{n/2+1}\frac{n!}{\prod_{i=1}^{n/2} i^2} = \frac{1}{n/2+1}\binom{n}{n/2} \tag{5.2}$$

(Note: this is the Catalan number $C_{n/2}$ [38].). Since each irrep has dimension at least 1, (5.2) is a lower bound (though nowhere close to a tight lower bound!) for the number of entries below the main diagonal in the Schur transform, and (5.2) is not polynomial in scaling. This is a relief, since

if we could acheive a polynomial sequence length by direct decomposition of the transform, all of our work above would be to no end. It is also easy to verify from our implementation that the Schur transform would not scale as a polynomial of n if we were to decompose it directly:

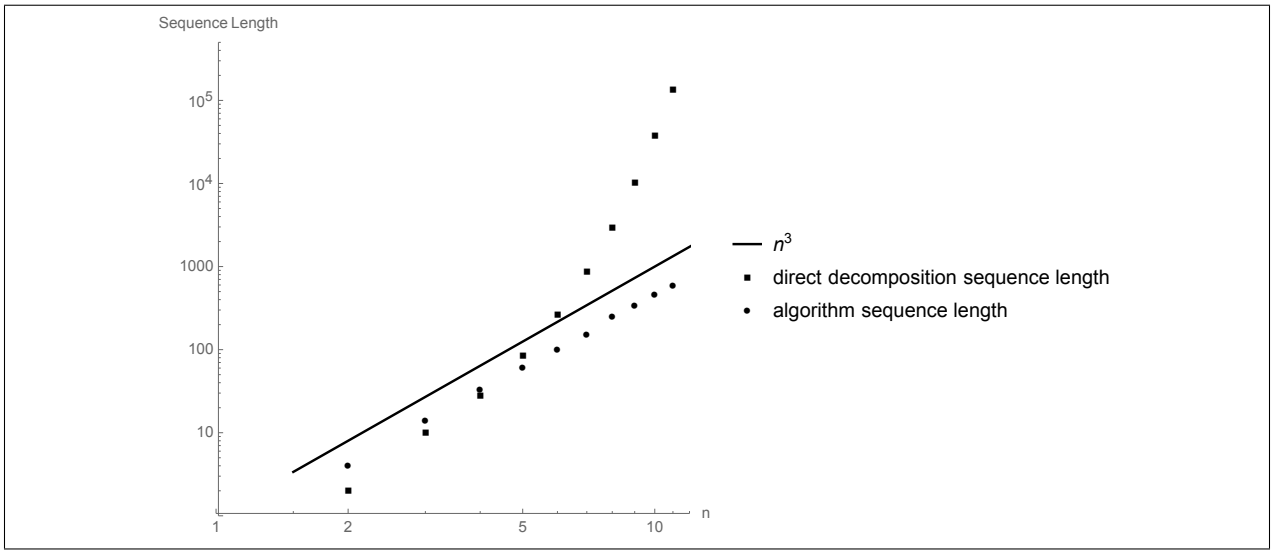


Figure 5.1: Two-level gate sequence lengths for qubits

Algorithm for qudits

A natural extension of our algorithm is to generalize from qubits to qudits of dimension d. With this in mind, all of our discussion was for general dimension until we began the actual analysis and implementation (beginning in §4.3). This included the abstract discussion of the structure of the algorithm, in which the symmetrized ordering discussed in §4.2 was the key piece. So up to this point, we can apply the same construction to qudits as we did to qubits: namely, we can decompose the Schur transform on n qudits into a product of Givens rotations in the same way. The only differences will be in the dimensions, numbers, and multiplicities of the irreps, and in the simple CG transforms used. We can analyze this decomposition, but first we will discuss the limitations of this approach.

The similarity of our qudit construction to our qubit construction ends with the part of the decomposition we spent the least time discussing: the decomposition of the Givens rotations into primitives from a universal gate set. The Clifford+T gate set is specific to qubits, so in order to complete this piece of the construction, we need a different universal gate set that applies to general qudits. Ideally, this gate set would be fault-tolerant, and the decomposition of a Givens rotation into primitives from the gate set would be $O(n \log(1/\epsilon))$. To our knowledge, no constructive algorithm that satisfies these conditions exists yet. Gottesman showed that fault-tolerant computation with qudits is possible, and provided a generalization of the Clifford+T set that is universal and fault-tolerant [39]. Thus, we can apply the Solovay-Kitaev algorithm ([40] and [41]) to this set (since it applies to any universal gate set) to obtain a fault-tolerant decomposition of any Givens rotation into $O(n \log^p(1/\epsilon))$ primitives, for $p \approx 3.97$. This will be sufficient to show that our con-

struction gives a efficient decomposition of the Schur transform into primitives; the only further improvements would be in the dependence on ϵ, for which p could in principle be decreased to 1 (note: this has been done ([42]), but not fault-tolerantly).

We can break down the number of Givens rotations required to construct our algorithm for the Schur transform on n qudits (dimension d) as follows:

- The Schur transform on n qudits requires $n-1$ super-CG transforms: from the super-CG transform that adds 1 qudit to 1 qudit to the super-CG transform that adds 1 qudit to $n-1$ qudits.
- The super-CG transform that adds 1 qudit to k qudits is block diagonal, with each block corresponding to the addition of 1 qudit to each irrep of k qudits. In particular, for $\mu = (\mu_1, \mu_2, ..., \mu_d) \vdash k$, the block associated to the addition of 1 qudit to $\mathcal{Q}_\mu^d$ has side length equal to d times the dimension of $\mathcal{Q}_\mu^d$. Note that μ cannot have degree greater than d, so we can write $\mu = (\mu_1, \mu_2, ..., \mu_d)$ as long as we remember that μ_2 through μ_d may be 0.
- The dimension of $\mathcal{Q}_\mu^d$ is given by the number of semistandard μ-tableaux t with entries in $\{1, 2, ..., d\}$ (see theorem 2.5.1). This number is given by a special case of *Stanley's hook-content formula*, (Thm. 15.3 in [43]), calculated as follows:

  For a partition μ, let the *content* of the box (i, j) be $i - j$. For the hook length as defined above:

  $$\#\text{ of semistandard tableaux with entries in } \{1, 2, ..., d\} = \prod_{\text{all boxes}} \frac{d + \text{content of box}}{\text{hook length of box}} \quad (5.3)$$

  For $\mu \vdash n$, μ has the maximal number of semistandard tableaux if $\mu = (n)$, as was the case for qubits. The number of semistandard tableaux with shape (n) and entries in $\{1, 2, ..., d\}$ is bounded by $(n+1)^{d-1}$, since we choose the location of the first 2, the first 3,... , and the first d in the tableau, each out of at most n possible locations.
- The number of distinct irreps is given by the number of distinct partitions of n with order bounded by d. This number is bounded by n^d, since we choose the length of each of the d rows from at most n possibilities.
- Our super-CG transform on k qudits will have a block corresponding to each distinct irrep of k qudits. The side length of each of these blocks is d times the dimension of the corresponding irrep. Since the dimension of any irrep is bounded by $(k+1)^{d-1}$ for k qudits, the total number of entries in the corresponding block is bounded by $d^2(k+1)^{2d-2}$. Thus, since the number of distinct irreps is bounded by k^d, and hence also by $(k+1)^d$, the number of nonzero entries in the super-CG is bounded by

  $$d^2(k+1)^{3d-2} \quad (5.4)$$

  This is a bound on the number of Givens rotations to decompose the super-CG transform.

- To build the Schur transform, we take the product of super-CG transforms on k qudits for k from 1 to $n-1$, so the total number of Givens rotations to decompose the Schur transform on n qudits is bounded by

$$d^2 \sum_{k=1}^{n-1} (k+1)^{3d-2} \leq d^2 \int_{k=1}^{n} (k+1)^{3d-2} dk < d(n+1)^{3d-1} < O(dn^{3d-1}) \tag{5.5}$$

  As discussed above, each Givens rotation can be decomposed to accuracy δ into $O(n \log^p(1/\delta))$ fault-tolerant primitives (for $p \approx 3.97$), so the Schur transform can be decomposed into a product of fault-tolerant primitives whose length is

$$O\left(d\, n^{3d} \log^p\left(\frac{dn^{3d-1}}{\epsilon}\right)\right) = O\left(d^{1+p} n^{3d} \log^p\left(\frac{dn}{\epsilon}\right)\right) \tag{5.6}$$

  for overall error bounded by ϵ. Since this is polynomial in n (with an additional logarithmic factor), our goal is achieved.

A tighter analysis than we have given is certainly possible, but ours is relatively simple and sufficient for proof of principle.

A final thought: our approach to building the qubit algorithm suggests a more general approach to quantum algorithm design for qubits. This involves only a restatement of points we have already made:

- We can implement a Givens rotation on n qubits with $O(n \log(1/\epsilon))$ Clifford+T operators, to accuracy ϵ.
- Any $d \times d$ unitary matrix can be decomposed into $\frac{d(d-1)}{2}$ Givens rotations. Thus the number of Givens rotations to decompose the unitary scales as d^2, the area of the unitary.
- Suppose we want to implement some $2^n \times 2^n$ unitary operator U. Further, suppose we can decompose U into a product of block diagonal unitaries U_j (these may be of varying dimensions, tensored into the appropriate identity matrices) such that the sum of the areas of the non-identity blocks is polynomial in n. Then decomposing each block in each of the U_j separately into Givens rotations and then decomposing these into Clifford+T gates will give us a sequence of Clifford+T gates whose length is polynomial in n, and whose product approximates U to accuracy ϵ. This works because each block in each U_j is itself a unitary matrix, so that the number of Givens rotations required to decompose it scales with its area. Thus the sum of the areas is proportional to the number of Givens rotations required to decompose the U_j, and to convert these to Clifford+T gates simply multiplies the sequence length by an additional factor of $n \log(1/\epsilon)$.
- In more explicitly mathematical terms:
  - for U a unitary operator on n qubits...

- if U can be decomposed into a product of block-diagonal unitaries U_j such that...
- the dimensions of the non-identity blocks in U_j are d_{jk} such that...
- $\sum_j \sum_k d_{jk}^2$ is polynomial in n...
- ...then U can be decomposed, by the method outlined above, into a product of Clifford+T gates whose length is polynomial in n.

This may be thought of as a sketch of a blueprint for a class of unitaries that can be efficiently implemented on a quantum computer.

Appendix A

Auxiliary Proofs

Theorem 2.2.1 (Maschke's Theorem [31], [32])**.** *Any finite group G is reductive.*

Proof. For any reducible G-module V, let W be a nontrivial G-submodule of V. Let $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle'$ be some inner product on V. Define a new inner product

$$\langle \mathbf{v}_1, \mathbf{v}_2 \rangle = \sum_{g \in G} \langle g\mathbf{v}_1, g\mathbf{v}_2 \rangle' \tag{A.1}$$

Note that for any $f \in G$,

$$\langle f\mathbf{v}_1, f\mathbf{v}_2 \rangle = \sum_{g \in G} \langle fg\mathbf{v}_1, fg\mathbf{v}_2 \rangle' = \sum_{h \in G} \langle h\mathbf{v}_1, h\mathbf{v}_2 \rangle' = \langle \mathbf{v}_1, \mathbf{v}_2 \rangle \tag{A.2}$$

so $\langle \cdot, \cdot \rangle$ is invariant under the action of G. Now consider

$$W^{\perp} = \{\mathbf{v} \in V \text{ such that } \langle \mathbf{v}, \mathbf{w} \rangle = 0 \; \forall \; \mathbf{w} \in W\} \tag{A.3}$$

We recall from linear algebra that $W^{\perp}$ is a subspace of V. Further, for any $\mathbf{w} \in W$ and any $\mathbf{u} \in W^{\perp}$, we have

$$\langle g\mathbf{u}, \mathbf{w} \rangle = \langle g^{-1}g\mathbf{u}, g^{-1}\mathbf{w} \rangle = \langle \mathbf{u}, g^{-1}\mathbf{w} \rangle = 0 \tag{A.4}$$

where the first step follows because $\langle \cdot, \cdot \rangle$ is invariant under the action of G. Thus, $W^{\perp}$ is also a G-submodule of V. From linear algebra, we know that for any subspace W of a vector space of V,

$$V = W \oplus W^{\perp} \tag{A.5}$$

and this completes our proof. □

Corollary 2.2.4.1. *Let V be an irreducible G-module, with $R : G \to GL(V)$ the corresponding irrep. Then the only operators in $GL(V)$ that commute with $R(g)$ for all $g \in G$ are operators of the form $c\boldsymbol{I}$, where $c \in \mathbb{C}$, and $\boldsymbol{I}$ is the identity operator.*

Proof. Let $T \in GL(V)$ such that

$$TR(g) = R(g)T \tag{A.6}$$

for all $g \in G$. Then

$$(T - c\mathbf{I})R(g) = R(g)(T - c\mathbf{I}) \tag{A.7}$$

for any $c \in \mathbb{C}$. (A.7) is a case of Schur's Lemma, where $W = V$ and $(T - c\mathbf{I}) : V \to V$ is the G-homomorphism. Consider the case where c is an eigenvalue of T. Then there exists some $\mathbf{v} \in V$ such that $T(d\mathbf{v}) = cd\mathbf{v}$ for any $d \in \mathbb{C}$. But then $(T - c\mathbf{I})(d\mathbf{v}) = \mathbf{0}$ for any $d \in \mathbb{C}$. Therefore $(T - c\mathbf{I})$ is not invertible, and thus is not an isomorphism, so it must be the zero map. Thus

$$T - c\mathbf{I} = 0 \quad \Rightarrow \quad T = c\mathbf{I} \tag{A.8}$$

□

Theorem 2.2.5. *For G-modules V and W with V irreducible, the multiplicity of V in W is given by the the dimension of the G-homomorphism space from V to W, i.e., by*

$$\textit{dim } Hom_G(V, W) \tag{A.9}$$

Proof. Let V, W be reductive G-modules, with V irreducible. Write W as a direct sum of its inequivalent irreps $W^{(i)}$ (we lose no generality here by assuming that we have a basis for W such that (2.10) is an equality). We will here write out the multiplicities of the irreps explicitly, so that the $W^{(i)}$ in the following equation are inequivalent: thus the version of (2.10) that we use here is

$$W = \bigoplus_i \mathcal{M}_i \otimes W^{(i)} \tag{A.10}$$

where $\mathcal{M}_i$ is the multiplicity space of $W^{(i)}$. We can simplify this by adopting the following shorthand: for m_i the dimension of $\mathcal{M}_i$ (and thus the multiplicity of $W^{(i)}$),

$$m_i W^{(i)} \equiv \mathcal{M}_i \otimes W^{(i)} \tag{A.11}$$

so that (A.10) becomes

$$W = \bigoplus_i m_i W^{(i)} \tag{A.12}$$

indicating that $W^{(i)}$ has multiplicity m_i.

Consider $\phi \in Hom_G(V, W)$. For any $\mathbf{v} \in V$ and $\mathbf{w}^{(i)} \in W^{(i)}$ such that $\phi(\mathbf{v}) = \mathbf{w}^{(i)}$,

$$\phi(g\mathbf{v}) = g\phi(\mathbf{v}) = g\mathbf{w}^{(i)} \in W^{(i)} \tag{A.13}$$

But we know that for any $\mathbf{u} \in V$, we can write $\mathbf{u} = gg^{-1}\mathbf{u}$, so let $\mathbf{v} = g^{-1}\mathbf{u}$ for $\mathbf{v}$ as defined above and some g. Plugging this into (A.13) gives

$$\phi(\mathbf{u}) = \phi(g\mathbf{v}) \in W^{(i)} \tag{A.14}$$

so since this holds for any $\mathbf{u} \in V$, we know that $\phi : V \to W^{(i)}$.

Now, by Schur's lemma (corollary 2.2.4.1), since $W^{(i)}$ are irreps, $\phi : V \to W^{(i)}$ must be the zero map or an isomorphism (let us rename this map ϕ_i, for clarity). Since $W^{(i)}$ are inequivalent, and since isomorphism is transitive, $\phi_i : V \to W^{(i)}$ can be an isomorphism for at most one value of i: call it k. We consider some ϕ such that ϕ_k is an isomorphism. Then $c\phi_k$ is also an isomorphism for any $c \in \mathbb{C}$. This now covers the zero map case as well. But the value of c can be different for each repetition of $W^{(k)}$ in the direct sum, so we can write $\phi : V \to \bigoplus_{j=1}^{m_k} W^{(k)}$ as

$$\phi = \bigoplus_{j=1}^{m_k} c_j \phi_k \tag{A.15}$$

That is,

$$\{E_{l,l} \otimes \phi_k | l \in [1, m_k] \subset \mathbb{Z}\} \tag{A.16}$$
$$\text{for } E_{l,l} \text{ the matrix of all zeros except for a 1 in entry } (l, l),$$

is a basis for $Hom(V, \bigoplus_{j=1}^{m_k} W^{(k)})$. Thus, (A.16) is isomorphic to a basis for $Hom(V, W)$, since the maps $\phi : V \to W^{(i)}$ are zero maps for all $i \neq k$. Therefore, $dim(Hom(V, W)) = m_k$, which is the multiplicity of V in W. □

Lemma 2.2.2. *For representations $R_1 : G \to GL(V_1)$ and $R_2 : G \to GL(V_2)$,*

$$Hom(V_1, V_2) \overset{G}{\cong} V_1^* \otimes V_2 \tag{A.17}$$

Proof. Let $X : G \to GL(Hom(V_1, V_2))$ be the representation defined by

$$X(g)M := R_2(g) M R_1(g) \tag{A.18}$$

and let $Y : G \to GL(V_1^* \otimes V_2)$ be the representation defined by

$$Y(g)(\langle u| \otimes |v\rangle) := \langle u|R_1^{-1}(g) \otimes R_2(g)|v\rangle \tag{A.19}$$

Let $M \in Hom(V_1, V_2)$,

$$M = \sum_i |v_i\rangle\langle i| \tag{A.20}$$

and let $U : Hom(V_1, V_2) \to V_1^* \otimes V_2$ be defined by

$$UM = U \sum_i |v_i\rangle\langle i| = \sum_i \langle i| \otimes |v_i\rangle \tag{A.21}$$

Then

$$UX(g)M = U \sum_i R_2(g)|v_i\rangle\langle i|R_1^{-1}(g) = \sum_i \langle i|R_1^{-1}(g) \otimes R_2(g)|v_i\rangle \tag{A.22}$$

and

$$Y(g)UM = Y(g)\sum_i \langle i| \otimes |v_i\rangle = \sum_i \langle i|R_1^{-1}(g) \otimes R_2(g)|v_i\rangle \tag{A.23}$$

Thus

$$Y(g) = UX(g)U^\dagger \tag{A.24}$$

□

Lemma 2.5.1.1. *Let $\lambda \vdash n$, and let T be a standard λ-tableau. Then if $d <$ deg(λ), support$(\Pi_T) = \{\boldsymbol{0}\}$.*

Proof. Let λ have degree greater than d, and let T be a standard λ-tableau. Let $r \in \text{Row}(T)$, and let $\mathbf{v}$ be a computational basis vector in $(\mathbb{C}^d)^{\otimes n}$ (i.e., $\mathbf{v} = |v_1 v_2 \cdots v_n\rangle$). Then since λ has degree greater than d, the first column of T contains more than d entries, so for some pair of the entries in the first column of T, the corresponding values in $r\mathbf{v}$ are the same.

For example, if the first column of T contains $1, 3, 5$ and $d = 2$, then at least some pair of the first, third, and fifth values in $r\mathbf{v}$ must be the same, since all three values are taken from the set $\{0, 1\}$.

For the general case, let i and j be the entries in the first column of T whose corresponding values in $r\mathbf{v}$ are the same. Then for any $c \in \text{Col}(T)$, $c(ij) \in \text{Col}(T)$ as well. Thus there is a subset $k \subset \text{Col}(T)$ that contains half of the elements in Col(T), such that Col$(T) = k \cup k(ij)$. Therefore, we can rewrite the sum in the column part of the Young symmetrizer as

$$\sum_{c \in \text{Col}(T)} \text{sgn}(c)c = \sum_{c \in k} \text{sgn}(c)c(\epsilon - (ij)) \tag{A.25}$$

But since the values at positions i and j in $r\mathbf{v}$ are the same, $(ij)r\mathbf{v} = r\mathbf{v}$, so

$$\left(\sum_{c \in \text{Col}(T)} \text{sgn}(c)c\right) r\mathbf{v} = \left(\sum_{c \in k} \text{sgn}(c)c(\epsilon - (ij))\right) r\mathbf{v} = \sum_{c \in k} \text{sgn}(c)c(r\mathbf{v} - (ij)r\mathbf{v}) = 0 \tag{A.26}$$

Since this holds for any $r \in \text{Row}(T)$,

$$\Pi_T(\mathbf{v}) = \left(\sum_{c \in \text{Col}(T)} \text{sgn}(c)c\right)\left(\sum_{r \in \text{Row}(T)} r\right)\mathbf{v} = \sum_{r \in \text{Row}(T)} \left(\sum_{c \in \text{Col}(T)} \text{sgn}(c)c\right) r\mathbf{v} = 0 \tag{A.27}$$

□

Lemma A. *Let $A_1, A_2, ..., A_m$ be a sequence of unitary operators. If $B_1, B_2, ..., B_m$ is a sequence of unitary operators such that B_j approximates A_j with error δ in the trace norm for all j, then the product $\Pi_{j=1}^m B_j$ approximates the product $\Pi_{j=1}^m A_j$ with error $\epsilon \leq m\delta$.*

Proof. First, consider two arbitrary unitaries A_1 and A_2 of dimension k. Suppose A_1 is approximated by B_1 with error ϵ_1, and A_2 is approximated by B_2 with error ϵ_2. As noted previously, these errors are calculated by the trace norm, scaled by the dimension. To simplify our expressions, define $C_1 = A_1 - B_1$ and $C_2 = A_2 - B_2$. Then we have

$$\epsilon_1 = \sqrt{\frac{1}{k}\text{Tr}\left(C_1^\dagger C_1\right)}, \quad \epsilon_2 = \sqrt{\frac{1}{k}\text{Tr}\left(C_2^\dagger C_2\right)} \tag{A.28}$$

Further, define $D_1 = \epsilon_1^{-1} C_1$ and $D_2 = \epsilon_2^{-1} C_2$. Then

$$\sqrt{\frac{1}{k}\text{Tr}\left(D_1^\dagger D_1\right)} = \sqrt{\frac{1}{k}\text{Tr}\left(\epsilon_1^{-1} C_1^\dagger \epsilon_1^{-1} C_1\right)} = 1 \tag{A.29}$$

$$\sqrt{\frac{1}{k}\text{Tr}\left(D_2^\dagger D_2\right)} = \sqrt{\frac{1}{k}\text{Tr}\left(\epsilon_2^{-1} C_2^\dagger \epsilon_2^{-1} C_2\right)} = 1 \tag{A.30}$$

Thus, D_1 and D_2 have norm 1. We want to find the error ϵ for $B_1 B_2$ as an approximation of $A_1 A_2$: for $C = A_1 A_2 - B_1 B_2$, we have

$$\epsilon = \sqrt{\frac{1}{k}\text{Tr}\left(C^\dagger C\right)} \tag{A.31}$$

We can expand C as follows:

$$C = A_1 A_2 - B_1 B_2 = A_1 A_2 - (A_1 - \epsilon_1 D_1)(A_2 - \epsilon_2 D_2) = \epsilon_2 A_1 D_2 + \epsilon_1 A_1 A_2 - \epsilon_1 \epsilon_2 D_1 D_2 \tag{A.32}$$

Thus, when we expand $C^\dagger C$, we will obtain 9 terms, each of which is a matrix of norm 1, scaled by some factors of ϵ_1 and ϵ_2 (for example, the first term is $\epsilon_2^2 A_1 D_2 A_1 D_2$). When we take the trace of this sum, the trace of each term is bounded by its scaling factor times k, the dimension, so

$$\frac{1}{k}\text{Tr}\left(C^\dagger C\right) \leq \epsilon_2^2 + \epsilon_1\epsilon_2 - \epsilon_1\epsilon_2^2 + \epsilon_1\epsilon_2 + \epsilon_2^2 - \epsilon_1^2\epsilon_2 - \epsilon_1\epsilon_2^2 - \epsilon_1^2\epsilon_2 + \epsilon_1^2\epsilon_2^2 \tag{A.33}$$

As long as $\epsilon_1 \leq 1$ and $\epsilon_2 \leq 1$, we can simplify further:

$$\frac{1}{k}\text{Tr}\left(C^\dagger C\right) \leq \epsilon_2^2 + 2\epsilon_1\epsilon_2 + \epsilon_2^2 = (\epsilon_1 + \epsilon_2)^2 \tag{A.34}$$

Thus, we have

$$\epsilon \leq \sqrt{(\epsilon_1 + \epsilon_2)^2} = \epsilon_1 + \epsilon_2 \tag{A.35}$$

By induction, if we take a product of m matrices that are approximated to error δ, the overall error will be bounded by $m\delta$. □

Appendix B

Efficient Compilation of a Unitary Operator

In this section, we describe an algorithm to approximately compile an arbitrary unitary operator into a product of one- and two-qubit gates taken from the Clifford+T gate set. This algorithm is nearly maximally efficient: it is within a logarithmic factor in the dimension of the unitary of the theoretical best efficiency. The Clifford+T gates are a commonly-used universal gate set that can be implemented fault-tolerantly. For background on universality, Clifford+T, and fault-tolerant quantum computing, see [27], [14], [15], and [40]. The algorithm we present here is only slightly modified from the one presented in [47] and achieves the same efficiency. We use the modified algorithm because, in our opinion, it is easier to visualize. The Clifford+T gates are

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, \qquad S = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix}, \qquad T = \begin{pmatrix}1 & 0\\ 0 & \frac{1}{\sqrt{2}}(1+i)\end{pmatrix},$$
$$CNOT = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix} \tag{B.1}$$

In [44], Harrow *et al.* showed that for any universal gate set, approximating a general unitary operator of dimension j to accuracy ϵ (in the Haar measure: see [45] and [46]) requires a sequence of primitives (gates from the universal set) of length

$$O(j^2 \log(1/\epsilon)) \tag{B.2}$$

We use the term *efficiency*, in describing a quantum algorithm, to refer to such a sequence length: thus (B.2) represents a theoretical bound on efficiency.

The algorithm presented here has efficiency

$$O(\log(j) j^2 \log(1/\epsilon)) \tag{B.3}$$

In describing and analyzing the algorithm, it will be easier to think of this efficiency as a function of the number of qubits n: since $j = 2^n$, we can rewrite the theoretical efficiency bound and our efficiency as

$$\text{theoretical efficiency bound: } O(4^n \log(1/\epsilon)), \qquad \text{our efficiency: } O(n \cdot 4^n \log(1/\epsilon)) \tag{B.4}$$

We break-down the algorithm as follows:

- The QR decomposition is used to factor a $d \times d$ nonsingular complex matrix into a product QR, where Q is a $j \times j$ unitary matrix, and R is a $j \times j$ upper triangular matrix [48]. There are numerous methods of implementing a QR decomposition: the one we will use is based on two-level rotations called Givens rotations. Two-level rotations are unitary operators that only act on two basis vectors in a given basis: thus they are isomorphic to 2×2 unitary operators. For example,

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 1 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{B.5}$$

  is a two-level rotation that is isomorphic to the Hadamard gate $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.

  To diagonalize a nonsingular matrix $A = (a_{ij})$ using Givens rotations, we iterate over the entries in A that are below the main diagonal: A_{ij} for $i < j$. Since the rows of A are linearly independent, each such a_{ij} can be zeroed by one Givens rotation, so in total we apply $m = \frac{j(j-1)}{2}$ Givens rotations to reduce A to upper triangular:

$$G_1 G_2 \cdots G_m A = R \tag{B.6}$$

  for $G_1, G_2, ..., G_m$ the sequence of Givens rotations, and R some upper triangular matrix. But then since the G_j are unitary, we have

$$A = G_m^\dagger G_{m-1}^\dagger \cdots G_1^\dagger R \tag{B.7}$$

  Now consider the case when $A = U$, a unitary operator. Then since the rows and columns of U are orthonormal and the application of the Givens rotations will preserve this orthonormality (since the Givens rotations are also unitary), when U is reduced to upper triangular it must in fact be diagonal, with diagonal entries of magnitude 1. By applying a one-level phase operation to each main diagonal entry, can rotate all of the diagonal entries to 1, to obtain the identity matrix. We can think of these phases as just one more two-level operation per diagonal entry. Thus, when we invert the decomposition as in (B.7), we obtain

$$U = G_m^\dagger G_{m-1}^\dagger \cdots G_1^\dagger \tag{B.8}$$

  where we now include the main diagonal phases in the sequence of G_j, so that $m = \frac{j(j+1)}{2}$: thus we have decomposed U into a product of $\frac{j(j+1)}{2}$ two-level rotations.

- We can now map each two-level rotation to a 2×2 unitary operator, using the construction presented on pp. 189-194 in [27]. This operation requires $O(n)$ Clifford+T gates: our extra factor of N in (B.4) comes from this.
- Finally, decompose each 2×2 (single-qubit) unitary into a product of $O(\log(1/\epsilon))$ Clifford+T gates with accuracy ϵ, using one of several known methods ([36] or [37], for example).

Thus in the first step we decompose U into

$$\frac{j(j+1)}{2} = \frac{2^n(2^n+1)}{2} = O(4^n) \tag{B.9}$$

Givens rotations. In the second step we map each Givens rotation to a single-qubit operator, using $O(n)$ Clifford+T gates. In the last step, we approximate each single-qubit operator to accuracy ϵ, using $O(\log(1/\epsilon))$ Clifford+T gates. Since ϵ can be made arbitrarily small, and for any constant k, $\log(k/\epsilon) = \log(1/\epsilon) + \log(k)$, we can approximate U to accuracy ϵ without changing the factor of $\log(1/\epsilon)$ in the efficiency. Thus taking the product of the sequence lengths for the above steps gives us our overall efficiency, $O(4^n n \log(1/\epsilon))$.

Appendix C

Implementation of Schur Transform Compiler

William M. Kirby
2017 Senior Honors Thesis in Physics
Advisor: Frederick W. Strauch
Williams College, Williamstown, MA, USA 01267

Algorithm for efficient compilation of the Schur transform on n qubits into a product of two-level gates.

Clebsch-Gordan Transform

lop[*d*] returns the lowering operator $J_- = J_{1-} + J_{2-}$ that acts on a composite system comprising a subsystem with total spin j_1 (corresponding to d dimensions), and a subsystem with total spin $j_2 = \frac{1}{2}$ (corresponding to 2 dimensions). Note: output vectors will not be normalized.

In[1]:=
```
lop[d_] := Module[{j1, m1, evec = Table[0, {i, 1, d}], lop1 = {}, lop2},
   (* Get effective total spin for d-dimensional system. *)
   j1 = (d - 1) / 2;

   (* First row of lowering operator is all zeros. *)
   AppendTo[lop1, evec];

   (* Get remaining rows of
    lowering operator for dimension d (total spin j1). *)
   Do[
    (* Get effective spin projection for index i in d-
     dimensions (total spin j1). *)
    m1 = j1 + 1 - i;

    AppendTo[lop1, ReplacePart[evec, i → Sqrt[j1 (j1 + 1) - m1 (m1 - 1)]]];,
    {i, 1, d - 1}
   ];

   (* Spin-1/2 lowering operator. *)
   lop2 = {{0, 0}, {1, 0}};

   Return[KroneckerProduct[lop1, IdentityMatrix[2]] +
     KroneckerProduct[IdentityMatrix[d], lop2]]
  ];
```

CG[*d*] returns the Clebsch-Gordan transform that adds 1 new qubit to a *d*-dimensional pre-symmetrized system (that is, the basis vectors for the *d*-dimensional system have definite values of m2, the total spin projection for the system).
Input to matrix is a vector in product basis form: indices correspond to |m1,m2⟩, ordered by m1, then m2.
Output from matrix is a vector in standard symmetrized form: indices correspond to |j,m⟩, ordered by j, then m.

In[2]:=
```
CG[d_] :=
  Module[{j1, evec = Table[0, {i, 1, 2 d}], cvec, mat = {}, lopd, a, b},
   If[d == 1,

    mat = {{1, 0}, {0, 1}};,

    (* Effective total spin of qubit is j2=
     1/2. Get effective total spin for d-dimensional system. *)
    j1 = (d - 1) / 2;

    (* New total dimension is 2d. evec is an empty vector of dimension 2d. *)
    (* Get first output basis vector with spin j1+1/2:
            |j=j1+1/2,m=j1+1/2⟩ = |m1=j1,m2=1/2⟩ *)
    cvec = ReplacePart[evec, 1 → 1];
    AppendTo[mat, cvec];

    (* Lowering operator. *)
    lopd = lop[d];

    (* Get remaining output basis vectors with spin j1+1/2. *)
    Do[
     cvec = FullSimplify[lopd.cvec / Norm[lopd.cvec]];
     AppendTo[mat, cvec];,
     {i, 1, d}
    ];

    (* Get first output basis vector with spin j1-1/2:
                must be a linear combination of |m1=j1,m2=-1/2⟩ and |m1=j1-1,
       m2=1/2⟩ that is orthogonal to the
      second output basis vector with spin j1+1/2 *)
    cvec = ReplacePart[evec, {2 → mat[[2, 3]], 3 → -mat[[2, 2]]}];
    AppendTo[mat, cvec];
```

```
(* Get remaining output basis vectors with spin j1-1/2. *)
Do[
 cvec = FullSimplify[lopd.cvec / Norm[lopd.cvec]];
 AppendTo[mat, cvec];,
 {i, 1, d - 2}
];
];

Return[mat]
];
```

Encode the information necessary to build a Schur transform out of successive Clebsch-Gordan transforms as follows:

- A set of qubits called ***seq*** that encodes the set of standard tableaux whose shapes are order-2 partitions of *n* (the number of qubits). For each partition *p*, ***seq*** keeps track of the multiplicity of the irrep associated to *p* by labeling the copies of it by standard tableaux.
- A set of qubits called ***par*** that encodes the set of order-2 partitions of *n*. These partitions label the irreps of the unitary group on *n* qubits.
- A set of qubits called ***stat*** that encodes the specific states within each irrep. The number of states in the irrep associated to partition *p* is given by the number of semistandard tableaux with shape *p* and entries in {1,2}.

CGBlocks[] returns the generalized Clebsch-Gordan transform (in block diagonal form) associated to the addition of 1 new qubit to *n* already-symmetrized qubits.

Assumes input vector is in the form
(symmetrized vector for *n* qubits, encoded with all necessary ancillary qubits for *n* already included) ⊗
(new computational qubit)

Output vector is in standard symmetrized form, pre reordering.

If *numbers* is True, returns the CG transform containing its actual values. If *numbers* is False, returns the CG transform showing only locations of nonzero values.

In[3]:=
```
CGBlocks[n_, nstat_, numbers_] :=
  Module[
   {pars = Table[{n - p2, p2}, {p2, 0, Floor[n / 2]}], npar, dstat, out, start, cg},
   (* Get dimensions of stat for each irrep (labeled by partitions): *)
   dstat = Table[p[[1]] - p[[2]] + 1, {p, pars}];

   If[n == 1, npar = nstat - 2, npar = nstat - 1];

   (* Initialize out: *)
   out = SparseArray[Table[{i, i} → 1, {i, 1, 2^(npar + nstat + 1)}]];

   Do[(* Iterate over distinct irreps: *)
    (* Upper left corner location: *)
    start = i 2^(nstat + 1);
    If[numbers,
     (* Get simple CG transform to be inserted: *)
     cg = SparseArray[CG[dstat[[i + 1]]]];
     Do[(* Build block (numbers): *)
      out[[start + k, start + l]] = cg[[k, l]];,
      {k, 1, 2 dstat[[i + 1]]},
      {l, 1, 2 dstat[[i + 1]]}
     ];,
     Do[(* Build block (no numbers): *)
       out[[k, l]] = 1;,
       {k, start + 1, start + 2 dstat[[i + 1]]},
       {l, start + 1, start + 2 dstat[[i + 1]]}
      ];
    ];,
    {i, 0, Length[pars] - 1}
   ];

   Return[out];
  ];
```

CGRearrange[] returns the permutation matrix that reorders the output of a *CGBlocks[]* transform by the new irreps (for *n*+1 qubits).

Assumes input vector is the output of the associated *CGBlocks[]* transform.

Output is in the form *{matrix, locations}*. *locations* is a list whose elements have the form *{inputlocation, outputlocation, dimension}* with one element per output irrep: *inputlocation* indicates the location index of the input irrep corresponding to that output irrep, *outputlocation* indicates the location index of the output irrep, and *dimension* indicates the dimension of the output irrep.

In[4]:=
```
CGRearrange[n_, nstat_] :=
  Module[{pars = Table[{n - p2, p2}, {p2, 0, Floor[n / 2]}], npar,
    dstat, xstart, ystart, m = Table[0, {i, 0, Floor[(n + 1) / 2] + 1}],
    swaps = {}, missingx = {}, missingy = {}, locations = {}},
   (* Get dimensions of stat for each irrep (labeled by partitions): *)
   dstat = Table[p[[1]] - p[[2]] + 1, {p, pars}];

   If[n == 1, npar = nstat - 2, npar = nstat - 1];

   Do[(* Iterate over distinct irreps for n: *)

    (* Move the first output irrep
      appearing in simple CG with input irrep pars[[i+1]]: *)
    (* Start location for block (x coordinate): *)
    xstart = i 2^(nstat + 1);
    (* Start location for block (y coordinate): *)
    ystart = m[[i + 1]] 2^(npar + nstat) + i 2^(nstat);
    (* m is a list that contains the numbers of each
     output irrep that have been moved to their correct locations
     so far: about to move top output irrep from input pars[[i+1]],
    so increment at [[i+1]]: *)
    m[[i + 1]] = m[[i + 1]] + 1;
    (* Add permutation entries to swaps list: *)
    Do[
     AppendTo[swaps, {ystart + k, xstart + k}],
     {k, 1, dstat[[i + 1]] + 1}
    ];
    (* Append upper-
     left corner location and dimension of block to locations: *)
    AppendTo[locations, {i 2^(nstat + 1), ystart, dstat[[i + 1]] + 1}];

    (* Move the second output irrep
      appearing in simple CG with input irrep pars[[i+1]]: *)
    xstart = i 2^(nstat + 1) + dstat[[i + 1]] + 1;
    ystart = m[[i + 2]] 2^(npar + nstat) + (i + 1) 2^(nstat);
```

```
  (* About to move bottom output irrep from input pars[[i+1]],
  so increment m at [[i+2]]: *)
  m[[i + 2]] = m[[i + 2]] + 1;
  (* Add permutation entries to swaps list: *)
  Do[
   AppendTo[swaps, {ystart + k, xstart + k}],
   {k, 1, dstat[[i + 1]] - 1}
  ];
  (* Append upper-
   left corner location and dimension of block to locations: *)
  AppendTo[locations, {i 2^(nstat + 1), ystart, dstat[[i + 1]] - 1}];,

  {i, 0, Length[pars] - 1}
 ];

 (* Fill in possible diagonal elements: *)
 Do[
  If[! (MemberQ[Flatten[swaps], i]),
    AppendTo[swaps, {i, i}];
   ];,
  {i, 1, 2^(npar + nstat + 1)}
 ];

 (* Reorder remaining indices to unused locations: *)
 Do[
  If[! MemberQ[Table[swaps[[j, 1]], {j, 1, Length[swaps]}], i],
    AppendTo[missingx, i];
   ];,
  {i, 1, 2^(npar + nstat + 1)}
 ];
 Do[
  If[! MemberQ[Table[swaps[[j, 2]], {j, 1, Length[swaps]}], i],
    AppendTo[missingy, i];
   ];,
  {i, 1, 2^(npar + nstat + 1)}
 ];
 Do[
  AppendTo[swaps, {missingx[[i]], missingy[[i]]}],
  {i, 1, Length[missingx]}
 ];

 Return[{SparseArray[Table[swaps[[j]] → 1, {j, 1, Length[swaps]}]], locations}];
];
```

Schur Transform

SchurDecompPartial[n] returns the decomposition of the Schur transform on *n* qubits into a sequence of super-CG transforms. Output is a list whose elements have form *{top, matrix, bottom}*, where *top* gives the number of qubits to be tensored in above (before) *matrix*, and *bottom* gives the number of qubits to be tensored in below (after).

In[5]:=
```
SchurDecompPartial[n_] :=
  Module[{nstat, out},
   nstat = Floor[Log[2, n]] + 1;
   (* List of matrices whose product is the Schur
    transform: elements of list have form {top, matrix, bottom},
   where top is the number of qubits to be tensored in above (before) matrix,
   and bottom is the number of qubits to be tensored in below (after). *)
   out = Table[{Max[0, n - 2 - i], CGRearrange[n - i, nstat][[1]].
       CGBlocks[n - i, nstat, True], i - 1}, {i, 1, n - 1}];
   (*AppendTo[out,{0,CGRearrange[1,nstat][[1]].CGBlocks[1,nstat,True],n-2}];*)
   Return[out];
  ];
```

SchurMatrix[n] returns the Schur transform matrix for *n* qubits. For the Schur transform matrix itself, the input register should be in the form

(***seq*** qubits (ancillas))⊗(***par*** qubits (ancillas))⊗(***stat*** qubits (computational input))

The output register will have the same form.

In[6]:=
```
SchurMatrix[n_] :=
  Module[{list},
   list = SchurDecompPartial[n];
   Return[Apply[Dot,
      Table[KroneckerProduct[IdentityMatrix[2^list[[i]][[1]]], list[[i]][[2]],
        IdentityMatrix[2^list[[i]][[3]]]], {i, 1, n - 1}]] // FullSimplify]
  ];
```

Givens[mat] returns the direct decomposition of unitary *u* into a sequence of two-level (Givens) rotations. Output is a list of two-level rotations and one-level phases whose product is equal to *u*.

In[7]:=
```
Givens[u_] :=
  Module[{list = {}, v, d, b, c, g, ph},
   d = Length[u]; (* Dimension of goal unitary. *)
   v = u // N; (* Stores current goal matrix. *)
   Do[
    Do[
      If[v[[i, j]] ≠ 0,
        (* Build two-level rotation {{b,-c},{c,b}}, to zero tempmat[[i,j]]. *)
        b = v[[j, j]] / Sqrt[Abs[v[[i, j]]]^2 + Abs[v[[j, j]]]^2];
        c = -(v[[i, j]] / Sqrt[Abs[v[[i, j]]]^2 + Abs[v[[j, j]]]^2]);
        (* Insert two-level rotation into identity of dimension d. *)
        g = IdentityMatrix[d];
        g[[j, j]] = b;
        g[[i, j]] = c;
        g[[j, i]] = -c;
        g[[i, i]] = b;
        (* Add conjugate-transpose of Givens rotation to output list,
        and update tempmat. *)
        AppendTo[list, g†];
        v = g.v // Chop;
       ];,
      {j, 1, i - 1}
     ];,
    {i, 2, d}
   ];
   (* Correct phases of diagonal elements. *)
   Do[
    If[v[[i, i]] != 1,
      g = IdentityMatrix[d];
      g[[i, i]] = v[[i, i]]*;
      AppendTo[list, g†];
     ];,
    {i, 1, d}
   ];
   Return[list]
  ];
```

SchurDecomp[n] returns the decomposition of the Schur transform on *n* qubits into a sequence of two-level rotations. Returns a list of elements with form *{top, rotation, bottom}*, where *rotation* is some two-level rotation,

In[8]:=
```
SchurDecomp[n_] :=
  Module[{list, givens, out = {}},
   list = SchurDecompPartial[n];
   Do[
    givens = Givens[cg[[2]]];
    Do[
     AppendTo[out, {cg[[1]], g, cg[[3]]}],
     {g, givens}
    ];,
    {cg, list}
   ];
   Return[out]
  ];
```

Extract Computational Basis

SchurIndices[n] returns the a list of the rows used to encode the computational states in the Schur transform matrix (as output by *SchurMatrix[n]*).

In[9]:=
```
SchurIndices[n_] :=
  Module[{nstat, indiceslist, inputs, outputs = {}},
   nstat = Floor[Log[2, n]] + 1;

   indiceslist =
    Table[{Max[i - 2, 0], CGRearrange[i, nstat][[2]], n - 1 - i}, {i, 1, n - 1}];
   (* At this point, indiceslist has one element per super-
    CG transform in the decomposition of the Schur transform. Each of
     these elements is itself a list of form {top, irrepslist, bottom},
   and irrepslist is a list whose elements have form
    {inputlocation, outputlocation, dimension},
   as described above in the description of CGRearrange[]. *)

   Do[
    indiceslist[[i]] =
      {indiceslist[[i]][[1]], (2^indiceslist[[i]][[3]]) indiceslist[[i]][[2]]};,
    {i, 1, Length[indiceslist]}
   ];
   (* At this point,
   indiceslist has one element per super-CG transform in the decomposition
     of the Schur transform. Each of these elements is itself a list,
   whose elements have form {top, {x, y, dimension}},
```

```
where x is the column corresponding to an input irrep, y is the row
 corresponding to an output irrep that comes from that input irrep,
and dimension is the dimension of that output irrep. This list
 contains one element for each distinct output irrep. *)

Do[
 indiceslist[[i]] =
   Flatten[
    Table[
     Table[{l[[1]] + j (2^(n + 2 nstat - 1 - i)),
       l[[2]] + j (2^(n + 2 nstat - 1 - i)), l[[3]]}, {l, indiceslist[[i]][[2]]}],
     {j, 0, -1 + 2^indiceslist[[i]][[1]]}],
    1];,
 {i, 1, Length[indiceslist]}
];
(* At this point, indiceslist has one element per super-
 CG transform in the decomposition of the Schur transform. Each of these
  elements is itself a list, whose elements have form {x, y, dimension},
where x is the column corresponding to an input irrep,
y is the row corresponding to an output irrep that comes from
 that input irrep, and dimension is the dimension of that output
 irrep. This list contains one element for each output irrep
 (from the previous step, we have added the multiplicities of the irreps). *)

Do[(* Start from the list corresponding to the super-CG transform that takes
    2 qubits to 3 qubits and iterate to the list corresponding to the super-
   CG transform that takes n-1 qubits to n qubits... *)
 inputs = Table[indiceslist[[i - 1]][[j]][[2]],
   {j, 1, Length[indiceslist[[i - 1]]]}];
 outputs = {};
 (* For each element in the current super-CG list,
 check whether its input corresponds to an output from the previous
  iteration. If it does, append it to the outputs list for this iteration. *)
 Do[
  If[MemberQ[inputs, slot[[1]]],
   AppendTo[outputs, slot];
  ];,
  {slot, indiceslist[[i]]}
 ];
 (* Update indiceslist with
  the new (reduced) outputs list for this iteration. *)
 indiceslist[[i]] = outputs;,
 {i, 2, Length[indiceslist]}
];
```

```
    (* For the final output irrep indices,
    we only care about the outputs from the super-CG transform that takes n-
     1 qubits to n qubits. This corresponds to the last element in indiceslist,
    so we drop the others. Also, we no longer care about the input
     locations that led to these irreps, so drop those. *)
    indiceslist = Table[indiceslist[[-1]][[i]][[2 ;; 3]],
      {i, 1, Length[indiceslist[[-1]]]}];
    (* At this point, indiceslist has one element per output irrep of
     the Schur transform: elements have form {outputlocation, dimension},
    where outputlocation is the location index of the irrep and
     dimension is its dimension.  For our final indiceslist,
    we want the rows corresponding to every state in every output irrep,
    not just the start locations for each irrep. *)
    indiceslist = Flatten[
      Table[Table[l[[1]] + j, {j, 1, l[[2]]}], {l, indiceslist}],
      1
     ];
    (* Now indiceslist is in its final form: it is a list of the indices of every
       row corresponding to a state in an output irrep of the Schur transform. *)

    Return[indiceslist]
   ];
In[10]:= (* Returns 1 if x is nonzero, 0 if x is 0. *)
  nonzeroto1[x_] :=
    Module[{},
     If[x == 0,
      0,
      1
     ]
    ];
```

SchurIrreps[n, numbers] displays the Schur transform matrix for *n* qubits. If *numbers* is True, displays the actual numbers; otherwise, displays the locations of nonzero entries. The rows corresponding to computational states (irreps) are highlighted in light red; if *numbers* is False, nonzero elements in these rows are dark red.

In[11]:=
```
SchurIrreps[n_, numbers_] :=
  Module[{background, schur, rows},
   schur = SchurMatrix[n];
   rows = Table[0, {i, 1, Length[schur]}, {j, 1, Length[schur]}];

   If[numbers,
    background = Table[l → LightRed, {l, SchurIndices[n]}];
    Grid[schur, Frame → True,
     Background → {None, background}, ItemStyle → Tiny, ItemSize → Full],

    background = SchurIndices[n];
    Do[
     rows[[i, j]] = LightRed;,
     {i, background},
     {j, 1, Length[schur]}
    ];
    schur = Map[nonzeroto1, schur, {2}];
    ArrayPlot[schur + rows, ImageSize → Large]
   ]
  ];
```

In[12]:=
```
{SchurIrreps[2, True], SchurIrreps[3, True]}
```

Out[12]=

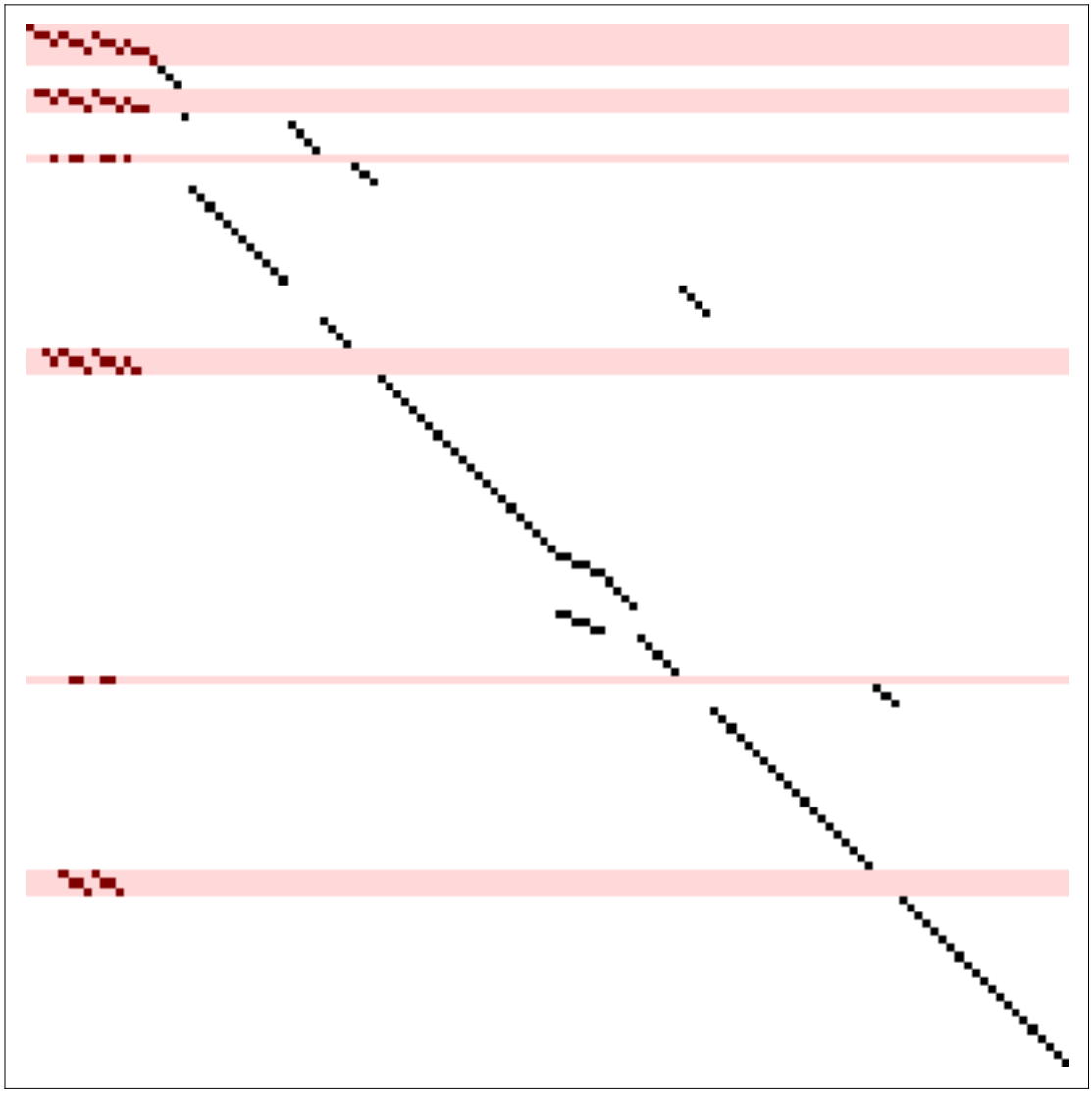
$$\left\{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{\frac{2}{3}} & -\frac{1}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{6}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{6}} & 0 & \frac{1}{\sqrt{6}} & -\sqrt{\frac{2}{3}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}\right\}$$

The matrix for *Schur[4, True]* (or larger) cannot be displayed on a single page. Instead, we show the matrix with the locations of the nonzero elements as well as the locations of the irreps marked. Rows

corresponding to irreps are highlighted in light red: nonzero elements in these rows are dark red. Nonzero elements in unused (ghost) rows are black.

In[13]:=
```
SchurIrreps[4, False]
```

Out[13]=



RestrictSchur[n] returns the Schur transform on *n* qubits, only including the computational input and output rows.

In[14]:=
```
RestrictSchur[n_] :=
  Module[{schur},
   schur = SchurMatrix[n];
   Table[schur[[i]][[1 ;; 2^n]], {i, SchurIndices[n]}]
  ];
```

In[15]:=
```
RestrictSchur[2] // MatrixForm
```

Out[15]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \end{pmatrix}$$

In[16]:=
```
RestrictSchur[3] // MatrixForm
```

Out[16]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \sqrt{\frac{2}{3}} & -\frac{1}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{6}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{6}} & 0 & \frac{1}{\sqrt{6}} & -\sqrt{\frac{2}{3}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 \end{pmatrix}$$

In[17]:=
```
Style[RestrictSchur[4] // MatrixForm, Small]
```

Out[17]=

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{6}} & 0 & \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & 0 & 0 & \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & 0 & \frac{1}{\sqrt{6}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \frac{\sqrt{3}}{2} & -\frac{1}{2\sqrt{3}} & 0 & -\frac{1}{2\sqrt{3}} & 0 & 0 & 0 & -\frac{1}{2\sqrt{3}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{6}} & 0 & \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & 0 & 0 & \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{6}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2\sqrt{3}} & 0 & 0 & 0 & \frac{1}{2\sqrt{3}} & 0 & \frac{1}{2\sqrt{3}} & -\frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & \sqrt{\frac{2}{3}} & 0 & -\frac{1}{\sqrt{6}} & 0 & 0 & 0 & -\frac{1}{\sqrt{6}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{3}} & 0 & -\frac{1}{2\sqrt{3}} & \frac{1}{2\sqrt{3}} & 0 & 0 & -\frac{1}{2\sqrt{3}} & \frac{1}{2\sqrt{3}} & 0 & -\frac{1}{\sqrt{3}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{6}} & 0 & 0 & 0 & \frac{1}{\sqrt{6}} & 0 & -\sqrt{\frac{2}{3}} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{3}} & 0 & -\frac{1}{2\sqrt{3}} & -\frac{1}{2\sqrt{3}} & 0 & 0 & -\frac{1}{2\sqrt{3}} & -\frac{1}{2\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Analysis

In[18]:=
```
(* Returns the two-
 level gate sequence length for our decomposition of the Schur transform. *)
Do[
 Print[{n, Length[SchurDecomp[n]]}],
 {n, 2, 2}
]
```

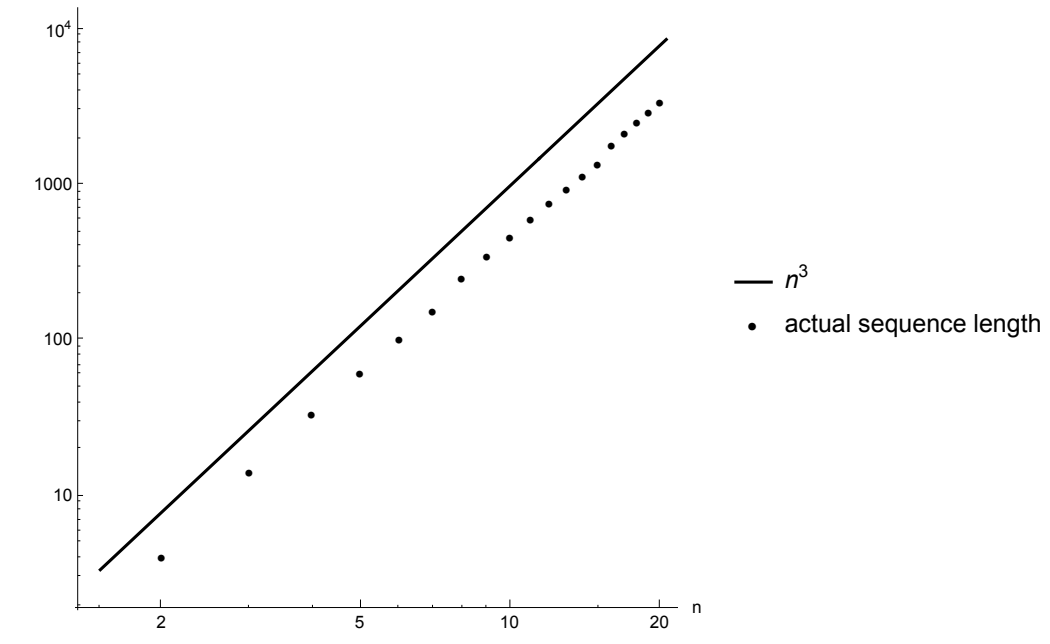
```
{2, 4}
```

In[19]:=
```
(* Elements have form {n, length},
where length is the sequence length (number of two-level rotations)
 for the decomposition of the Schur transform on n qubits. *)
data = {{2, 4}, {3, 14}, {4, 33}, {5, 61}, {6, 101}, {7, 151}, {8, 248},
   {9, 342}, {10, 456}, {11, 592}, {12, 747}, {13, 922}, {14, 1119},
   {15, 1339}, {16, 1780}, {17, 2116}, {18, 2494}, {19, 2907}, {20, 3357}};
```

In[20]:=
```
(* Points on an n^4 curve (the theoretical upper bound): *)
thrdata = Table[{n, n^3}, {n, 1.5, 20.5, 0.1}];
```

In[21]:=
```
ListLogLogPlot[{thrdata, data}, AspectRatio → 1,
 AxesLabel → {"n", "Sequence Length"}, PlotStyle → Black,
 Joined → {True, False}, PlotMarkers → {"", •},
 PlotLegends → Placed[{"n^3", "actual sequence length"}, Right]]
```

Out[21]=



Comparison to direct decomposition of the Schur transform into two-level gates:

In[22]:=
```
(* Returns the length of the decomposition of u into Givens rotations: *)
GivensCount[u_] :=
  Module[{l = 0, v, d, b, c, g, ph},
   d = Length[u]; (* Dimension of goal unitary. *)
   v = u // N; (* Stores current goal matrix. *)
   Do[
    Do[
      If[v[[i, j]] ≠ 0,
        (* Build two-level rotation {{b,-c},{c,b}}, to zero v[[i,j]]: *)
        b = v[[j, j]] / Sqrt[Abs[v[[i, j]]]^2 + Abs[v[[j, j]]]^2];
        c = -(v[[i, j]] / Sqrt[Abs[v[[i, j]]]^2 + Abs[v[[j, j]]]^2]);
        (* Insert two-level rotation into identity matrix of dimension d: *)
        g = IdentityMatrix[d];
        g[[j, j]] = b;
        g[[i, j]] = c;
        g[[j, i]] = -c;
        g[[i, i]] = b;
        (* Count matrix and and update v: *)
        l = l + 1;
        v = g.v // Chop;
       ];,
      {j, 1, i - 1}
     ];,
    {i, 2, d}
   ];
   (* Count phase corrections: *)
   Do[
    If[v[[i, i]] != 1,
      l = l + 1;
     ];,
    {i, 1, d}
   ];
   Return[l]
  ];
```

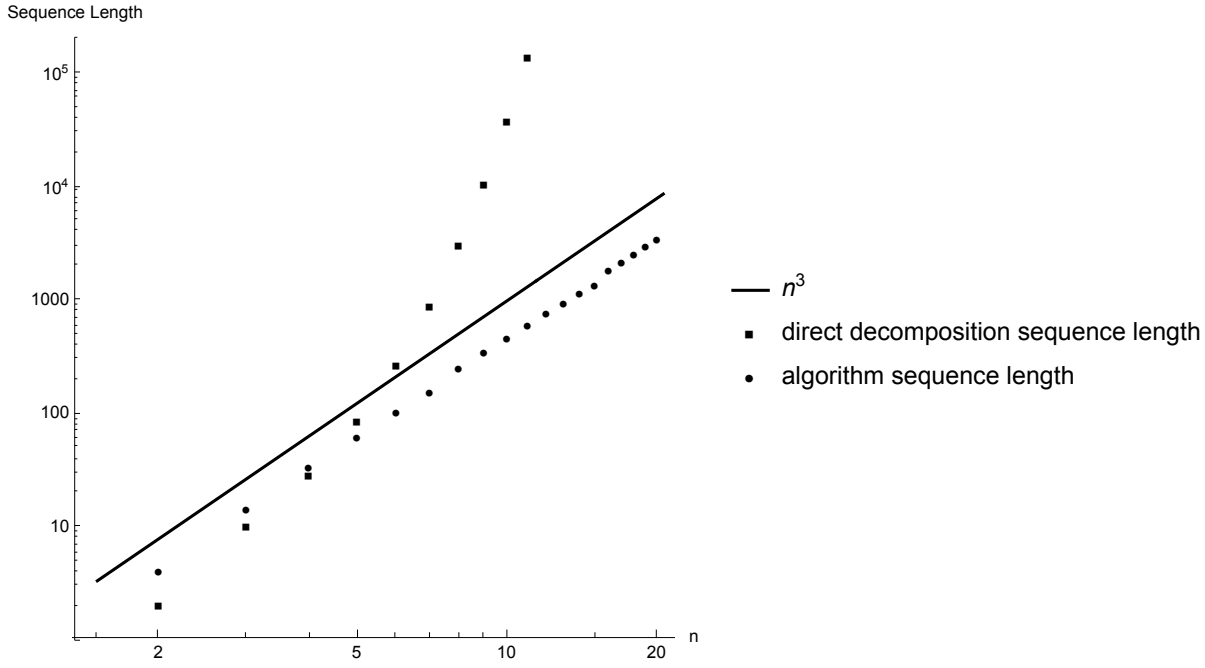
In[23]:=
```
(* Returns the sequence length for a direct
  decomposition of the Schur transform into two-level gates. *)
Do[
 Print[{n, GivensCount[RestrictSchur[n]]}],
 {n, 2, 2}
]
```

```
{2, 2}
```

In[24]:=
```
datadirect = {{2, 2}, {3, 10}, {4, 28}, {5, 84}, {6, 261},
   {7, 858}, {8, 2977}, {9, 10 365}, {10, 37 191}, {11, 135 226}};
```

In[25]:=
```
ListLogLogPlot[{thrdata, datadirect, data}, PlotRange → {1, 2 × 10^5},
 AspectRatio → 1, AxesLabel → {"n", "Sequence Length"}, PlotStyle → Black,
 Joined → {True, False, False}, PlotMarkers → {"", ■, •}, PlotLegends → Placed[{"n^3",
    "direct decomposition sequence length", "algorithm sequence length"}, Right]]
```

Out[25]=



Spin Operator Diagonalization

A quick sanity check: the Schur transform on *n* qubits ought to diagonalize the spin operators on *n* qubits...

In[26]:=
```
(* Spin projection operator on n qubits: *)
sz[n_] := Sum[KroneckerProduct[IdentityMatrix[2^(i - 1)],
    (1 / 2) PauliMatrix[3], IdentityMatrix[2^(n - i)]], {i, 1, n}];
```

In[27]:=
```
(* Total spin operator on n qubits: *)
s[n_] := Sum[MatrixPower[Sum[KroneckerProduct[IdentityMatrix[2^(i - 1)],
      (1 / 2) PauliMatrix[j], IdentityMatrix[2^(n - i)]], {i, 1, n}], 2], {j, 1, 3}];
```

Check:

n = 2 :

In[28]:=
```
RestrictSchur[2].s[2].RestrictSchur[2]† // FullSimplify // MatrixForm
```

Out[28]//MatrixForm=

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

In[29]:=
```
RestrictSchur[2].sz[2].RestrictSchur[2]† // FullSimplify // MatrixForm
```

Out[29]//MatrixForm=

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

n = 3 :

In[30]:=
```
RestrictSchur[3].s[3].RestrictSchur[3]† // FullSimplify // MatrixForm
```

Out[30]//MatrixForm=

$$\begin{pmatrix} \frac{15}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{15}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{15}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{15}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{3}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{3}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{3}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{3}{4} \end{pmatrix}$$

In[31]:=
```
RestrictSchur[3].sz[3].RestrictSchur[3]† // FullSimplify // MatrixForm
```

Out[31]//MatrixForm=

$$\begin{pmatrix} \frac{3}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{3}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} \end{pmatrix}$$

```
n = 4 :
```

In[32]:=
```
RestrictSchur[4].s[4].RestrictSchur[4]† // FullSimplify // MatrixForm
```

Out[32]//MatrixForm=

$$\begin{pmatrix}
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

In[33]:=
```
RestrictSchur[4].sz[4].RestrictSchur[4]† // FullSimplify // MatrixForm
```

Out[33]//MatrixForm=

$$\begin{pmatrix}
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Bibliography

[1] D. Bacon, I. L. Chuang, and A. W. Harrow, Proc. 18th Ann. ACM-SIAM SODA pp. 1235–1244 (2007).

[2] P. Zanardi and M. Rasetti, Phys. Rev. Lett. **79**, 3306 (1997).

[3] D. A. Lidar, I. L. Chuang, and K. B. Whaley, Phys. Rev. Lett. **81**, 2594 (1998).

[4] C. A. Bishop and M. S. Byrd, J. Phys. A: Math. Theor. **42**, 055301 (2009).

[5] K. Audenaert, M. Nussbaum, A. Szkola, and F. Verstraete, Commun. Math. Phys. **279**, 251 (2008).

[6] M. Hayashi, J. Phys. A: Math. Theor. **35**, 10759 (2002).

[7] M. Keyl and R. F. Werner, Phys. Rev. A **64**, 052311 (2001).

[8] R. D. Gill and S. Massar, Phys. Rev. A **61**, 042312 (2002).

[9] M. Hayashi and K. Matsumoto, *Universal distortion-free entanglement concentration* (2002).

[10] S. D. Bartlett, T. Rudolph, and R. W. Spekkens, Phys. Rev. Lett. **91**, 027901 (2003).

[11] D. Bacon, I. L. Chuang, and A. W. Harrow, Phys. Rev. Lett. **97**, 170502 (2006).

[12] S. J. Berg, Ph.D. thesis, University of California (2012).

[13] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra* (SIAM, 2000).

[14] D. Gottesman, Phys. Rev. A **57**, 127 (1998).

[15] S. Bravyi and A. Kitaev, Phys. Rev. A **71**, 022316 (2005).

[16] R. P. Feynman, Int. J. Theor. Phys **21**, 467 (1982).

[17] D. Deutsch, Proc. Roy. Soc. Lond. A **400**, 97 (1985).

[18] P. W. Shor, SIAM J. Sci. Statist. Comput. **26**, 1484 (1997).

[19] L. K. Grover, Proc. 28th Annual ACM Symposium on the Theory of Computing pp. 212–219 (1996).

[20] D. Bacon, A. M. Childs, and W. van Dam, Proc. 46th IEEE Symposium on Foundations of Computer Science pp. 469–478 (2007).

[21] E. Knill, R. Laflamme, and G. J. Milburn, Nature **409** (2001).

[22] C. R. Monroe and D. J. Wineland, Scientific American pp. 64–71 (2008).

[23] J. Clarke and F. K. Wilhelm, Nature **453**, 1031 (2008).

[24] D. D. Awschalom, R. Epstein, and R. Hanson, Scientific American pp. 84–91 (2007).

[25] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, Nature **536**, 63 (2016).

[26] B. Lekitsch, S. Weidt, A. G. Fowler, K. Mølmer, S. J. Devitt, C. Wunderlich, and W. K. Hensinger, Science Advances **3**, e1601540 (2017).

[27] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, UK, 2001).

[28] J. S. Townsend, *A Modern Approach to Quantum Mechanics* (University Science Books, Mill Valley, California, 2012), 2nd ed.

[29] J. Sakurai, *Modern Quantum Mechanics* (Addison-Wesley Publishing Company, Inc., USA, 1985).

[30] R. Goodman and N. R. Wallach, *Representations and Invariants of the Classical Groups*, vol. 68 of *Encyclopedia of Mathematics and Its Applications* (Cambridge University Press, Cambridge, UK, 1998).

[31] H. Maschke, Math. Ann. (German) **50**, 492 (1898).

[32] H. Maschke, Math. Ann. (German) **52**, 363 (1899).

[33] B. E. Sagan, *The Symmetric Group, Representations, Combinatorial Algorithms, and Symmetric Functions*, no. 203 in Graduate Texts in Mathematics (Springer-Verlag New York, 2001), 2nd ed.

[34] K. Schertler and M. H. Thoma, Annalen Phys. **5**, 103 (1996).

[35] P. G. Burke, *Clebsch-Gordan and Racah Coefficients* (Springer International Publishing AG., 2010), vol. 61 of *Springer Series on Atomic, Optical, and Plasma Physics*, chap. R-Matrix Theory of Atomic Collisions, pp. 607–617.

[36] P. Selinger, *Efficient clifford+t approximation of single-qubit operators*.

[37] V. Kliuchnikov, D. Maslov, and M. Mosca, Phys. Rev. Lett. **110**, 190502 (2013).

[38] T. Koshy, *Catalan Numbers with Applications* (Oxford University Press, 2008).

[39] D. Gottesman, Chaos Solitons Fractals **10**, 1749 (1999).

[40] A. Y. Kitaev, A. H. Shen, and M. N. Vyalyi, *Classical and Quantum Computation, volume 47 of Graduate Studies in Mathematics* (American Mathematical Society, Providence, RI, 2002).

[41] C. M. Dawson and M. A. Nielsen, Quantum Information and Computation **0**, 1 (2005).

[42] D.-S. Wang and B. C. Sanders, New J. Phys. **17**, 043004 (2015).

[43] R. P. Stanley, Stud. Appl. Math. **50**, 259 (1971).

[44] A. W. Harrow, B. Recht, and I. L. Chuang, J. Math. Phys. **43**, 4445 (2002).

[45] A. Haar, Ann. Math. **34**, 147 (1933).

[46] L. Loomis, *Introduction to Abstract Harmonic Analysis* (D. Van Nostrand and Co., 1953).

[47] V. Kliuchnikov, *Synthesis of unitaries with clifford+t circuits*.

[48] D. S. Bernstein, *Matrix Mathematics* (Princeton University Press, 2005).