Representation Theory in Quantum Computing

A Thesis
Presented to
The Established Interdisciplinary Committee for Mathematical and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Dale D. Schandelmeier-Lynch

May 2025

Approved for the Committee
(Mathematics and Computer Science)

Jamie Pommershiem	Greg Anderson

Acknowledgements

I want to thank a few people.

Preface

This is an example of a thesis setup to use the reed thesis document class.

List of Abbreviations

You can always change the way your abbreviations are formatted. Play around with it yourself, use tables, or come to CUS if you'd like to change the way it looks. You can also completely remove this chapter if you have no need for a list of abbreviations. Here is an example of what this could look like:

ABC	American Broadcasting Company
CBS	Columbia Broadcasting System
CDC	Center for Disease Control
CIA	Central Intelligence Agency
CLBR	Center for Life Beyond Reed
CUS	Computer User Services
FBI	Federal Bureau of Investigation
NBC	National Broadcasting Corporation

Table of Contents

List of Tables

List of Figures

Abstract

The preface pretty much says it all.

Introduction

Theoretical quantum computing often employs a model of asymptotic analysis known as *oracle problems* as an abstraction to make comparisons between classical and quantum computers easier. An oracle is a theoretical function whose inner workings cannot be observed or manipulated; algorithms are only allowed to choose an input to the oracle and recieve its output. The goal when given an oracle problem is to create an algorithm which learns a given characteristic of the function hidden by the oracle. We usually assume that the function hidden by the oracle is sampled randomly from a distribution known to the algorithm. A simple example is as follows. For a function $f : \{a_0, a_1\} \to \mathbb{Z}/2\mathbb{Z}$ uniformly sampled from the space of possible functions, determine the mod 2 sum $a_0 \oplus a_1$. Here we define the oracle O to offer information to the algorithm through the black-box function $O(x) \coloneqq f(x)$. A classical algorithm would have to make 2 queries to O for exact learning, as it must query every element of the codomain to learn the values of a_o and a_1. As it turns out, a well known quantum algorithm called Deutsch's algorithm can learn the value of $a_0 \oplus a_1$ with certainty after just 1 query!

maybe this shouldn't be called a model

Also of interest in oracle problems is *bounded error* learning, where we consider how many queries are needed until an algorithm has some $\geq \frac{1}{2}$ chance of guessing the characteristic correctly. Contrast the previous example with a problem where the algorithm has to learn f exactly; before making any queries the classical algorithm has a $\frac{1}{2}$ of guessing $a_0 \oplus a_1$ correctly, and this doesn't change after making a query. On the other hand learning the exact function has a $\frac{1}{4}$ chance of success with 0 queries, and a $\frac{1}{2}$ chance of success with 1 query. If we considered an extreme version of this problem, $f : \{a_0, a_1\} \to \mathbb{Z}/1000\mathbb{Z}$, then we observe that a classical algorithm has a near-zero chance of guessing the function before it makes its final query. For most oracle problems, this pattern holds true classically; the algorithms knows nearly nothing about the function until it knows the whole function. Effectively, bounded error learning is equivalent to exact learning in this case. Shockingly, quantum algorithms have a different tendency. The function will be nearly unknown for the first few

queries, but after some asymptotically smaller number of queries than that needed for exact learning, nearly the whole function will be known! We are thus concerned with comparing the exact classical query complexity with both the exact and bounded error quantum query complexity of a given oracle problem. Of note is that we are proving we have found a neccesary and sufficent number of queries to solve the problem; we have an algorithm which solves the problem with the given asymptotics, and it is optimal, meaning no algorithm make asymptotically less queries.

Our oracle problem comes from [CP], as a specfication of the more general coset identification problem: suppose a group G acts by permutations on a finite set Ω (we call Ω a G-set). An algorithm is given access to an oracle which takes an element $\omega \in \Omega$ and returns $a \cdot \omega$ for some hidden group element (uniformly sampled) $a \in G$. The goal is to guess the hidden element $a \in G$. We will be concerned with the groups S_n and $A_n \leq S_n$ acting on the set of k element subsets of n, and on the set of regular partitions of n into b parts of size a.

To answer these problems we will use a crucial result from [CP] which enables us to determine optimal quantum query complexity with a correspondence. Taking the tensor product of a representation derived from our G-set Ω with itself repeatedly will be equivalent to making oracle queries, turning our quantum computing problem into one of representation theory. By understanding the representation corresponding to Ω and the way it tensors with itself, we will be able to find the query complexity. Because our G-sets have S_n and A_n as groups, we will be deeply concerned with the representation theory of S_n; combinatorial structures known as *Young diagrams* and *Young tableaux* will offer a useful lens through which we can analyze the representation theory.

Chapter 1

Background

We first introduce the quantum computing and representation theory neccesary to understand the problem statement in context.

1.1 Quantum Computing

Quantum computing utilizes the nonclassical nature of quantum mechanics to achieve asymptotic improvements over classical algorithms. Not all problems admit a quantum "speedup"; we will be more specific about our specific computational framework after the neccesary background. To illuminate the physical-mathematical framework in which quantum computing takes place, we will expound the postulates of quantum mechanics and the associated linear algebra from [NC][Chapter 2].

1.1.1 Quantum Mechanics

Due to theorem 4.2 from [CP], it turns out that our results rely almost entirely on representation theory; nevertheless we introduce the 4 fundemental postulates from chapter 2 of [NC] to provide context for the quantum model of computation. Throughout this section we use the "bra-ket" (or Dirac) notation to represent vectors in vector spaces. This is inherited from physics and uses the symbols $|\cdot\rangle$ called ket, $\langle\cdot|$ called bra, and $\langle\cdot|\cdot\rangle$ called braket (as we have "bracketed" the symbols);

$$|\psi\rangle$$

is a vector we have labeled ψ. In quantum computing we are concerned with complex vector spaces by postulate; we need to define inner products both for this postulate

and to elaborate on bra-ket notation.

Definition 1.1.1 (Inner product, [NC, 2.1.4])**.** A function $(\cdot,\cdot) : V \times V \to \mathbb{C}$ is a (complex) inner product if it satisfies the following requirements:

1. $(\cdot,\cdot)$ is linear in the second argument:

$$\left(|v\rangle, \sum_i \lambda_i |w_i\rangle\right) = \sum_i \lambda_i \left(|v\rangle, |w_i\rangle\right).$$

2. $\left(|v\rangle, |w\rangle\right) = \left(|v\rangle, |w\rangle\right)^*$.

3. $\left(|v\rangle, |v\rangle\right) \geq 0$ with equality if and and only if $|v\rangle = 0$.

The intuition of braket notation is that $\langle\phi|$ is the dual of the vector $|\phi\rangle$, which in the corresponding matrix presentation of vectors corresponds to turning a column vector into a row vector. Therefore taking the standard matrix product

$$\langle\phi|\,|\psi\rangle$$

is equivalent to the standard inner product $\left(|\phi\rangle, |\psi\rangle\right) = \sum_{k=1}^{n} \phi_k^* \psi_k$, which we notate as

$$\langle\phi|\psi\rangle;$$

here the symbol $\cdot^*$ represents the unary complex conjugation operation. The elegance of braket notation, apart from saving space, is that we can easily tell the output of complicated linear expressions; it easy to see that $\langle\phi|\psi\rangle\,\langle x|y\rangle$ equals a scalar, while it takes a little thinking to see that

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

equals a scalar.

We are now ready to introduce our first postulate. As mathematicians we needn't worry about how pysicists experimentally constructed these postulates, although their (approximate) truth will be neccesary if we are to physically realize quantum computers and employ the alogrithms we design. If we take these postulates at face value, phenomenon that are strange from a classical physics standpoint feel like natural consequences of the mathematics; in that sense it can be helpful to abstract away your physical intution and instead focus on the consequences of these postulates when we start doing computation.

Postulate 1.1.1 ([NC, 2.2.1]). Associated to any isolated physical system is a complex vector space with inner product (that is, a Hilbert space) known as the *state space* of the system. The system is completely described by its *state vector*, which is a unit vector in the system's state space.

Here unit vector refers to a vector with norm 1; the standard norm we employ is $|| \, |\psi\rangle \, || \coloneqq \sqrt{\langle\psi|\psi\rangle}$. Note that quantum mechanics doesn't tell us the state space or state vector of a particular system; it only describes in general the characteristics systems share. In our quantum computing model we regard these physical systems as memory and perform operations on this memory through the appropriate mathematical morphism, which in this case are linear operators. A minor detail is that Hilbert spaces have more properties when they are infinite dimensional, but in our case we will only ever use finite memory and so our Hilbert spaces are always finite dimensional.

The simplest quantum mechanical system is the *qubit*. A qubit resides in $\mathbb{C}^2$ and so has a two dimensional state space. We use $|0\rangle$ and $|1\rangle$ to label the elements of an orthonormal basis for the qubit; in practice we let

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Note that $|0\rangle$ is not the zero vector 0; we use $|0\rangle$ despite this overlap in reference to the two states of a bit, 0 and 1. What makes the qubit interesting in contrast is that we are not limited to state vectors $|0\rangle$ and $|1\rangle$; we know from linear algebrea that we can express any element of the state basis as

$$|\psi\rangle = a \, |0\rangle + b \, |1\rangle \, ,$$

where a and b are in the complex numbers. Even with the condition that $|\psi\rangle$ be a unit vector the state of the qubit is described by 4 real numbers (that is $x + iy$ where $x, y \in \mathbb{R}$ for both a and b) instead of one bit! In practice we can only physically measure 3 of these values, but nevertheless a qubit can take on far more values than a bit because it can exist in a superposition of the states $|0\rangle$ and $|1\rangle$; that is both a and b can be nonzero at the same time. In our case we will be using the state space $\mathbb{C}\Omega$ with standard basis $\{|\omega\rangle \, | \omega \in \Omega\}$; you can imagine we have defined an abelian group structure on Ω and so we are considering the group algebra $\mathbb{C}\Omega$ by linearization.

We are now ready for our next postulate:

Postulate 1.1.2 ([NC, 2.2.2])**.** The evolution of a *closed* quantum system is described by a *unitary transformation*. That is, the state $|\psi\rangle$ of the sysem at time t_1 is related to the state $|\psi'\rangle$ of the system at time t_2 by a unitary operator U which depends only on the times t_1 and t_2;

$$|\psi'\rangle = U|\psi\rangle .$$

With operators we can define the last piece of the bra-ket puzzle. If we consider the vector $|v\rangle$ from a Hilbert space V, and the vector $|w\rangle$ from a Hilbert space W, then we define the out product notation for a linear operator $|w\rangle\langle v| : V \to W$ as

$$(|w\rangle\langle v|)(|v'\rangle) \coloneqq |w\rangle\langle v|v'\rangle = \langle v|v'\rangle|w\rangle .$$

We can thus interpret the notation $|w\rangle\langle v|v'\rangle$ in two ways: as a linear operator fed $|v'\rangle$, and as the vector $|w\rangle$ scaled by the inner product of $\langle v|v'\rangle$. In terms of matrix multplication we see here the associativity property at work; the product of the column and row vectors $|w\rangle\langle v|$ creates a matrix which then acts as a linear operator on $|v'\rangle$, or the product of row and column vectors creates a scalar which then scales $|w\rangle$.

To define a unitary operator, we first need the *adjoint* or *Hermitian conjugate* [NC, 2.1.6] of an operator. This the unique operator $A^\dagger$ such that for all vectors $|v_1\rangle, |v_2\rangle \in V$,

$$(|v_1\rangle, A|v_2\rangle) = (A^\dagger|v_1\rangle, |v_2\rangle).$$

Here the operation is the inner product; by convention we define $(|v\rangle)^\dagger \coloneqq \langle v|$ so that $(A|v_1\rangle)^\dagger|v_2\rangle = \langle v_1|A^\dagger|v_2\rangle$. When we have a matrix representing an operator A, the standard definition of the adjoint is to take the conjugate and then the tranpose of A: $A^\dagger \coloneqq (A^*)^T$. When $A^\dagger = A$, we call A *Hermitian* or *self-adjoint*. As a broader class we say that an operator A is *normal* if $AA^\dagger = A^\dagger A$; it is immediate that Hermitian operators are normal. Finally we say an operator U is unitary if $U^\dagger U = I$, where I is the identity operator (or matrix); for finite vector spaces it can be proven that this implies $UU^\dagger = I$, so unitary operators are also normal. We care about unitary products because they preserve inner products; observe that

$$(U|v\rangle, U|w\rangle) = \langle v|U^\dagger U|w\rangle = \langle v|I|w\rangle = \langle v|w\rangle \qquad \text{[NC, 2.36]}.$$

Postulate 2 demands that operators be unitary so that the unit condition of the state vector is maintained.

We are now ready for postulate 3, which tells us what happens when an isolated

quantum system is measured. To measure a quantum system it must be exposed to an external physcial system, and so the rule of unitary evolution from postulate 2 no longer neccearily applies.

Postulate 1.1.3 ([NC, 2.2.3])**.** Quantum measurements are described by a collection $\{M_m\}$ of *measurement operators*. These are operators acting on the state space of the system being measured. The index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement then the probability that result m occurs is given by

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle\,,$$

and the state of the system after the measurement is

$$\frac{M_m\,|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}.$$

The measurement operators satisfy the *completeness equation*,

$$\sum_m M_m^\dagger M_m = I.$$

The completeness equation expresses the fact that probabilities sum to one:

$$1 = \sum_m p(m) = \sum_m \langle\psi|M_m^\dagger M_m|\psi\rangle\,.$$

The takeaways from this postulate are twofold. Measurements in a quantum system can change the state of the state vector; information can be lost by performing a measurement. From the real numbers contained in a state vector — 3 in the case of the qubit — we can only observe a finite number of states. In fact, if we are trying to distinguish between states that are not orthonormal, then we lose the guarentee of distinguishability. Because some state $|\psi\rangle$ is not orthonormal to some other distinict state $|\phi\rangle$, there is a component of $|\psi\rangle$ that is orthogonal to $|\phi\rangle$ and a component that is parallel to $|\phi\rangle$; therefore when you measure a system with state vector $|\psi\rangle$ there is a nonzero chance of it being misidentified as $|\phi\rangle$. This means that we can only perfectly distinguish between a number of states up to the dimension of the state space. Furthermore cascaded measurements $\{L_l\}$ and $\{M_m\}$ are equivalent to a single measurement with operators $\{N_{lm}\}$ where $N_{lm} \coloneqq M_m L_l$ [NC, 2.57], so if we don't care about the evolution of the system over time there is no reason to not take

all the measurements we want to at once.

This leads us to the concept of the POVM, or "positive operator-valued measure." If we perform a measurement on a state vector $|\psi\rangle$ with measurement operators M_m, then we know the probability of outcome m is given by $p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle$. Now if we define

$$E_m \coloneqq M_m^\dagger M_m,$$

we can conclude that $\{E_m\}$ is a set of positive operators such that $\sum_m E_m = I$ and $p(m) = \langle\psi|E_m|\psi\rangle$. Therefore this set is sufficent to determine the probability of each measurement outcome if we don't care about the post-measurement state [NC, 2.2.6]. In fact we can take this the other direction and define a POVM to be any set of operators such that each operator is positive and the completeness relation $\sum_m E_m = I$ is obeyed. This is legitimate because if we define measurements $M_m \coloneqq \sqrt{E_m}$, we see that $\sum_m M_m^\dagger M_m = \sum_m E_m = I$, so we have constructed a set of measurements $\{M_m\}$ with POVM $\{E_m\}$. We prefer POVMs to measures as a convience, because when we do computation we take one measurement at the end and discard the state vector.

Our last postulate tells us how to describe quantum systems made out of multiple distinct systems; this is useful in computation because we model adding memory as composing more physical systems.

Postulate 1.1.4 ([NC, 2.2.8])**.** The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if we have systems numbered 1 through n, and system number i is in the state $|\psi_i\rangle$, then the joint state of the total system is $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$.

A tensor product is an operation on vector spaces analagous to "multiplying" them; In the same way that a cartesian product is an operation on sets that changes what the elements of the resulting set look like (turns them into tuples), the tensor product of the spaces will change the vectors into "tensor products" in a different sense.

The following definitions are from [NC, 2.1.7]. Suppose we have hilbert spaces V and W — although tensor products can be defined on plain vector spaces — of dimension m and n respectively. Then $V \otimes W$ is an mn dimensional vector space. As an analogue for finite sets $X : |X| = m$ and $Y : |Y| = n$ the cartesian product has cardinality $|X \times Y| = mn$. The elements of this vector space are linear combinations of tensor products $|v\rangle \otimes |w\rangle$ where $|v\rangle \in V$ and $|w\rangle \in W$. The tensor product is akin to a tuple in that elements from the first vector space go in the first index, elements

from the second vector space go in the second index, et cetra. If $\{|i\rangle_n\}$ and $\{|j\rangle_n\}$ are (preferably but not neccesarily orthonormal) bases for the spaces V and W then $\{|i\rangle \otimes |j\rangle\}_{n,m}\}$ is a basis for $V \otimes W$. We abbreviate tensor products as $|v\rangle\,|w\rangle$ or $|vw\rangle$ for convinence. For example if V is a qubit then $V \otimes V$, also notated as $V^{\otimes 2}$ to indicate it has been tensored with itself (akin to squaring) contains the element $|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle = |00\rangle + |11\rangle$. We define the tensor product at the element (vector) level to have the following properties:

1. For an arbitrary scalar z and vectors $|v\rangle \in V$ and $|w\rangle \in W$,

$$z(|v\rangle \otimes |w\rangle) = (z\,|v\rangle) \otimes |w\rangle = |v\rangle \otimes (z\,|w\rangle).$$

2. For an arbitrary $|v_1\rangle\,, |v_2\rangle \in V$ and $|w\rangle \in W$,

$$(|v_1\rangle + |v_2\rangle) \otimes |w\rangle = |v_1\rangle \otimes |w\rangle + |v_2\rangle \otimes |w\rangle\,.$$

3. For an arbitrary $|v_1\rangle\,, |v_2\rangle \in V$ and $|w\rangle \in W$,

$$(|v_1\rangle + |v_2\rangle) \otimes |w\rangle = |v_1\rangle \otimes |w\rangle + |v_2\rangle \otimes |w\rangle\,.$$

4. For an arbitrary $|v\rangle \in V$ and $|w_1\rangle\,, |w_2\rangle \in W$,

$$|v\rangle \otimes (|w_1\rangle + |w_2\rangle) = |v\rangle \otimes |w_1\rangle + |v\rangle \otimes |w_2\rangle\,.$$

These distributivity laws intuitively make tensor products function like multiplication between vectors. Furthermore, we can define linear operators on tensor products which inherit from linear operators on the tensored spaces; let $|v\rangle \in V, |w\rangle \in W$, and let A and B be linear operators on V and W. Then we define the linear operator $A \otimes B$ in $V \otimes W$ by the equation

$$(A \otimes B)(|v\rangle \otimes |w\rangle) \coloneqq A\,|v\rangle \otimes B\,|w\rangle\,.$$

This definition of $A \otimes B$ now naturally extends linearly to all the elements of $V \otimes W$; that is

$$(A \otimes B)(\sum_i a_i\,|v_i\rangle \otimes |w_i\rangle) \coloneqq \sum_i a_i A\,|v_i\rangle \otimes B\,|w_i\rangle\,.$$

It can be shown that $A \otimes B$ thus defined is a well-defined linear operator. Finally a natural inner product can be inherited from the tensored spaces; we define this

product as

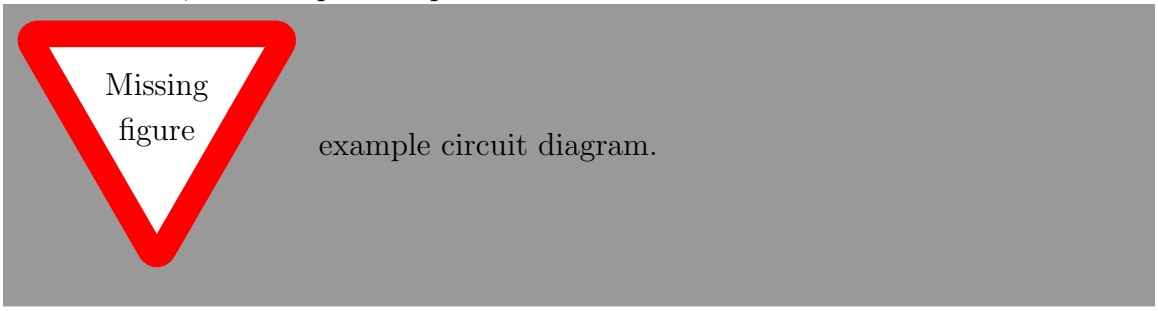
$$\left(\sum_i a_i \left|v_i\right\rangle \otimes \left|w_i\right\rangle, \sum_j b_j \left|v_j'\right\rangle \otimes \left|w_j'\right\rangle\right) := \sum_{i,j} a_i^* b_j \left\langle v_i | v_j'\right\rangle \left\langle w_i | w_j'\right\rangle .$$

This function can also be shown to be a well-defined inner product. Equiped with this inner product $V \otimes W$ is now an inner product/Hilbert space, and it inherits our previously established structure of an adjoint, unitarity, normality, Hermanicity, et cetra.

1.1.2 Our Computational Model

We have now covered all the postulates of quantum mechanics! The standard model of quantum computation is simple: we start with a state space $(\mathbb{C}^2)^{\otimes n} \otimes (\mathbb{C}^2)^{\otimes a}$ which corresponds to an input requiring n qubits to store and an extra workspace of a ancilla qubits — usually of constant size — needed as extra space for certain computations. We can assume that the memory begins in any desired state $|\psi\rangle$ as such a state can be initialized by applying a constant number of unitary operators. Unitary operators are then applied to do some kind of computation, and a POVM is taken to measure the output. Instead of worrying about the time or space complexity of this model, we will be concerned with query complexity; this simplifies complexity to the question of how many times a given black box function (the "oracle") is applied as a unitary operator. Because this model is non-uniform — the operators applied change as the input size changes — we describe a quantum algorithm as a recipe to construct the neccesary sequence of operations based on the input size. This model lends itself to representation as a circuit diagram, where reading from left to right we initialize qubits as seperate inputs, apply unitary operations to qubits, and then perform a measurement; an example is depicted below.

is this true? maybe only takes one?



example circuit diagram.

The model we employ in our research is similar, but instead uses the state space $\mathbb{C}\Omega \otimes \mathbb{C}^N$; here we are taking the group ring $\mathbb{C}\Omega$ where Ω is a G-set with cyclic group

structure as a single register. You can imagine it as a $|\Omega|$-dit because $\mathbb{C}\Omega$ has $|\omega\rangle \in \Omega$ as a basis, but endowed with extra structure because the action of G on Ω naturally corresponds to a group of unitary operators on $\mathbb{C}\Omega$. The space $\mathbb{C}^N$ is simply ancilla but regarded as one N-dit rather than multiple qubits. Note that the postfix "-dit" here indicates a register of a given dimension (d for dimension), whereas qubit refers to a register of dimension 2 (as "bi-" means 2). To understand why we use this model, we will need an introduction to representation theory.

1.2 Introductory Representation Theory

this is copy-pasted, needs to be modified to flow with rest of background.

Our results and model employ a field of abstract algebra known as representation theory, which is concerned with correspondences between groups and linear operators. By turning group elements into (unitary) linear operators, we will be able to use them as gates in our quantum computing model. Our introduction here will be focused on laying the groundwork for the representation theory of S_n, the subject in representation theory relevant to our results.

Our first definition will be specific to the set of invertible $d \times d$ matrices with entries in $\mathbb{C}$. This is the full complex matrix algebra, because we have a vector space of matrices scaled by the the complex numbers with a multiplication defined by matrix multiplication.

citation style in bibtex file is different here

Definition 1.2.1 ([Sag, Definition 1.2.1])**.** A *representation* (more specifically a *matrix representation*) of a group G is a group homomorphism

$$\rho : G \to \mathrm{GL}_d$$

where GL_d is the complex general linear group of degree d; that is, the set of invertible $d \times d$ matrices from $\mathbb{C}^d$ to $\mathbb{C}^d$.
To break down the definition of a group homomorphism, this means that

$$\rho(e) = I_d$$

where e is the identity of the group, and I_d is the identity matrix of degree d in $\mathbb{C}$, and

$$\rho(g)\rho(h) = \rho(gh) \text{ for all } g, h \in G.$$

This means that multiplying group elements and then taking their matrix representation is equivalent to taking their matrix representations and performing matrix

multiplication on them.

Note that representations can be defined for other fields like $\mathbb{R}$ or $\mathbb{Z}/P\mathbb{Z}$, and doing so changes the core theorems of the theory; we won't be concerned with this because our quantum mechanical postulates demand we use $\mathbb{C}$. A point of confusion with representations is that they are defined to be *homomorphisms*, not *monomorphisms*. We might intuitively imagine that a matrix representation is equivalent to the group just with matrices instead of group elements, but information/group elements can be lost by mapping nontrivial elements of the group to the identity matrix in the linear group (that is, a representation can have nontrivial kernel). Furthermore, additional representations can be created by choosing a different degree or performing a change of basis on a given representation. A small degree can possibly restrict the possible representations to exclude monomorphisms. On the other hand, a change of basis shouldn't fundementally change the linear transformation a matrix represents. We can consider representations ρ, ϕ equivalent (through an equivalence relation) if there exists an invertible matrix T such that $\rho(g) = T^{-1}\phi(g)T$ for all $g \in G$.

This relationship between matrices and linear transformations can be further used to avoid a choice of basis; we now introduce G-modules.

Definition 1.2.2 ([Sag, Definition 1.3.1])**.** Let V be a vector space and G be a group. Then V is a *G-module* if there is a group homomorphism

$$\rho : G \to \mathrm{GL}(V),$$

where $\mathrm{GL}(V)$ is the set of general linear transformations of the vector space V. Equivalently, V is a G-module if there is a multiplication $g \cdot v$ such that

1. $g \cdot v \in V$,
2. $g(cv + dw) = c(gv) + d(gw)$,
3. $(gh)v = g(hv)$, and
4. $ev = v$

for all $g, h \in G$, $v, w \in V$, and scalars $c, d \in \mathbb{C}$. That is, there is a left action $G \circlearrowright V$ on V that distributes over addition.

These two definitions are equivalent because acting by a group element g is equivalent to applying the linear transformation $\rho(g)$. G-modules and matrix representations are closely related. We use any matrix representation X with degree

matching the dimension of V to construct a G-module by defining the linear action $g \cdot v := X(g)v$. Likewise, if we have a G-module we can define a matrix representation X by choosing a basis B for the linear transformations $\rho(g)$ and then construct $X(g)$. An important difference between representations and modules is that we have a choice of vector space when making a module. We can use this fact (and the fact that actions are part of our module definition) to naturally represent arbitary group actions.

Example 1.2.3. We turn a group action $G \circlearrowright X$ into a G-module by first making the vectors of our space be formal linear sum of elements of the set: $[x] = \sum_{x_i \in X} \lambda_i x_i$. The group action then can then be linearly extended to act on $\mathbb{C}[X]$, the vector space generated by X over $\mathbb{C}$:

$$\begin{aligned} G &\circlearrowright \mathbb{C}[X] \\ (g, [x]) &\mapsto \sum_{x_i \in X} \lambda_i (g \cdot x_i) \end{aligned}$$

Now we can define a G-module

$$\rho : G \to \mathrm{GL}(|X|, \mathbb{C}[X])$$

where $\rho(g)$ is the linear transformation induced by the action of G on X; if we take the standard basis of $\mathbb{C}[X]$ to be the set X, then $\rho(g)$ can be constructed as a matrix which permutes each x_i according to the image of the action of g on each x_i.

We will be concerned with finding *irreducible* representations/modules, which are fundemental to understanding the representation theory of a group because for representations over $\mathbb{C}$ and similarly nice fields any representation can be represented as a product of irreducible representations. We will concern ourselves with irreducible modules primarily because they are easier to work with and are in close correspondence with irreducible representations. To do so we first need submodules.

Definition 1.2.4 ([Sag, Definition 1.4.1])**.** Let V be a G-module. a *submodule* of V is a subspace W that is closed under the action of G; that is,

$$w \in W \implies gw \in W \text{ for all } g \in G.$$

Example 1.2.5. Any G-module V has the submodules $W = V$ as well as $W = \{0\}$, where 0 is the zero vector. These two submodules are called trivial, and any other submodules are called nontrivial.

Definition 1.2.6 ([Sag, Definition 1.4.1])**.** A nonzero G-module V is *reducible* if it contains a non-trivial submodule W. Otherwise V is said to be reducible.

We will need the following definition in our analysis of S_n to demonstrate that we have found a complete set of irreducible non-isomorphic $G(S_n)$-modules.

Definition 1.2.7 ([Sag, Definition 1.6.1])**.** Let V and W be G-modules. Then a *G-homomorphism* is a linear transformation $\Theta : V \to W$ such that

$$\Theta(g \cdot v) = g \cdot \Theta(v)$$

for all $g \in G$ and $v \in V$.
A *G-isomorphism* is a bijective G-homomorphism.

Finally, we introduce a main theorem that we will need for the representation theory of S_n. This theorem that tells us that any representation can be constructed out of irreducible representations. To understand it we first need the notion of a direct sum.

Definition 1.2.8 ([Sag, Definition 1.5.1])**.** Let V be a vector space with subspaces U and W. Then V is the *(internal) direct sum of U and W*, written $V = U \oplus W$, if every $v \in V$ can be written uniquely as a sum

$$v = u + w, \quad u \in U, w \in W.$$

In the same way that the tensor product is like multiplication for vector spaces, the direct sum is analagous to the addition of vector spaces.

Theorem 1.2.9 ([Sag, Theorem 1.5.3]; Matchske's theorem)**.** *Let G be a finite group and let V be a nonzero G-module. Then*

$$V = W_1 \oplus W_2 \oplus \cdots \oplus W_k$$

where each W_i is an irreducible G-submodule of V.

We tack on one extra definition for the tensor product of representations; while we won't need it for the representation theory of the symmetric group, we will use it in our problem statement and our results.

Definition 1.2.10 ([Sag, Definition 1.11.1])**.** Let G and H have matrix representations X and Y,respectively.The tensor product representation, $X \otimes Y$, assigns to each

$(g, h) \in G \times H$ the matrix

$$(X \otimes Y(g, h) = X(g) \otimes Y(h).$$

1.2.1 Character Theory

It turns out that much of the information of a representation can be obtained from the trace of its corresponding matrices. This is shocking and useful because the information of a matrix with n^2 entries can be obtained from the sum of just n entries. We now define the character, which stores all the traces of a representation.

Definition 1.2.11 ([Sag, Definition 1.8.1])**.** Let $X(g)$ where $g \in G$ be a matrix representation. Then the *character* of X is

$$\chi g = \mathrm{tr}(X(g)),$$

where tr denotes the trace of a matrix. In other words χ is the function

$$\chi : G \to \mathbb{C}$$

where

$$g \mapsto \mathrm{tr}(X(g)).$$

If V is a G-module, then its character is the character of a matrix represention X corresponding to V. If matrices X and Y both correspond to V, then we know $Y = TXT^{-1}$ for some T. Thus, for all $g \in G$,

$$\mathrm{tr}(Y(g)) = \mathrm{tr}(TX(g)T^{-1}) = \mathrm{tr}(X(g)),$$

since trace is invariant under conugation. So X and Y have the same character and our notion of module character is well defined.

The following proposition describes the basic properties of a character:

Proposition 1.2.12 ([Sag, Proposition 1.8.5])**.** *Let X be a matrix representation of a group G of degree d with character χ.*

1. $\chi(\epsilon) = d$.
2. *If K is a conjugacy class of G then*

$$g, h \in K \implies \chi(g) = \chi(h).$$

3. *If Y is a representation of G with character ψ, then*

$$X \cong Y \implies \chi(g) = \psi(g)$$

for all $g \in G$.

Note that we proved 3 *in our definition of a character.*

This next proposition is similar in flavour and characterizes the relationship between irreducible representations and their corresponding group.

Proposition 1.2.13 ([Sag, Proposition 1.10.1])**.** *Let G be a finite group and suppose that the set V_i ranging over i forms a complete list of distinct irreducible G-modules. Then*

1. *$\sum_i (\dim V_i)^2 = |G|$, and*

2. *the number of V_i equals the number of conjugacy classes of G.*

1.3 Representation Theory of the Symmetric Group

1.3.1 Motivation

We will consider the symmetric group S_n and its representations; in doing so, we will find how these representations correspond to mathematical objects known as Young diagrams.

Definition 1.3.1. A *Young diagram* is pictoral presentation of a *integer partition*. We define an integer partition λ of an integer $n \in \mathbb{Z}$ to be a sequence of integers, indexed by λ_i, such that:

1. the sequence is (weakly) decreasing:

$$\forall i \in \mathbb{N}, \lambda_i \geq \lambda_{i+1}.$$

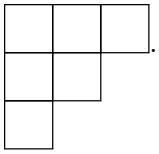
2. The sequence sums to n:

$$\sum_{i \in \mathbb{N}} \lambda_i = n.$$

3. And each index is nonnegative:

$$\forall i \in \mathbb{N}, \lambda_i \geq 0.$$

We usually leave out the trailing zeros of an integer partition, and denote an integer partiton of n with the notation $\lambda \vdash n$ or $(\lambda_1, \lambda_2, \cdots) \vdash n$. A Young diagram represents an integer partition as a picture by drawing λ_i boxes in the i-th row, from top to bottom. For example, the integer partition $(3, 2, 1) \vdash 6$ is represented by the Young diagram

 .

We state without proof that the number of irreducible representations (up to isomorphism) of a group is equal to the number of conjugacy classes in the group [Sag, Proposition 1.10.1] . It is a fundemental abstact algebra fact that conjugacy classes of S_n are in bijection with the cycle types of S_n, and because cycle types are unordered lists we can map each cycle type to its unique weakly decreasing ordering. This forms a bijection between cycle types of S_n and integer partitions of n. Composing bijections, we can conclude that each integer partition of n uniquely corresponds to an irreducible representation of S_n!

probably should at least write out the proposition here

Our goal will be to find these irreducible representations by creating their corresponding irreducible S_n-modules.

1.3.2 Tableaux and Tabloids

In order to use tableaux in S_n-modules, we'll need to define the action of S_n on a tableaux of a partition of size n.

Definition 1.3.2. A *Young Tableaux* is a numbering of a Young diagram of size n with the numbers $\{1 \cdots n\}$ with no repeats allowed.

There is no ordering condition, unlike with a further refinement we will use later called standard Young tableaux (hereafter refered to as SYT).

The action of $\sigma \in S_N$ on a tableaux T of size n is to replace the box i in T with the box $\sigma(i)$ in $\sigma \cdot T$.

Definition 1.3.3 ([Ful, Pg. 84])**.** The *row group* of T, denoted $R(T)$, is the set of permutations which permute the entries of each row among themselves. If $\lambda = (\lambda_1 \leq$

$\lambda_2 \leq \cdots \leq \lambda_k < 0$), then $R(T)$ is a product of symmetric groups $S_{\lambda_1} \times S_{\lambda_2} \times \cdots \times S_{\lambda_k}$. In fact, if row 1 of T contains the numbers $\{j_1 \cdots i_1\} \subseteq [n]$, row 2 contains $\{j_2 \cdots i_2\}$, et cetra, then $R(T)$ is the product of the permutations of each set — $S_{\{j_1 \cdots i_1\}} \times S_{\{j_2 \cdots i_2\}} \times \cdots \times S_{\{j_k \cdots i_k\}}$.

We analagously define the column group of T, $C(T)$, to be the set of permutations which permute the entries of the columns among themselves. This is equivalent to the row group of the transpose of the tableaux.

These subgroups of S_n are compatible with the action of S_n on T in the following way:

$$R(\sigma \cdot T) = \sigma \cdot R(T) \cdot \sigma^{-1} \text{ and } C(\sigma \cdot T) = \sigma \cdot C(T) \cdot \sigma^{-1}.$$

Example 1.3.4. For the tableaux below, $(143) \in R(T)$ but $(12) \notin R(T)$. Analagously $(143) \notin C(T)$ and $(12) \in C(T)$.

1	4	3
2	5	

If we perform the action $(12)(34) \cdot T$ we now see that $\sigma(143)\sigma^{-1} = (12)(34)(143)(12)(34) = (243) \in R(\sigma \cdot T)$. Intuitively conjugation is swapping elements of the underlying set undergoing a bijection.

2	3	4
1	5	

We now define tabloids, which will be used in our construction of S_n-modules.

Definition 1.3.5 ([Ful, Pg. 85])**.** A *tabloid* is an equivalence class of tableaux where two tableaux of the same shape are equivalent if their rows contain the same values. The tabloid containing T is denoted $\{T\}$. Two tableaux are in the same class ($\{T\} = \{T'\}$) when $T' = \sigma \cdot T$ for some $\sigma \in R(T)$.

Example 1.3.6. Tabloids are notated as tableaux without vertical lines to convey that changing the ordering of values within a row doesn't change the tabloid.

$$\begin{array}{|c c c|}\hline 1 & 4 & 3 \\ \hline 2 & 5 \\ \cline{1-2}\end{array} = \begin{array}{|c c c|}\hline 4 & 1 & 3 \\ \hline 2 & 5 \\ \cline{1-2}\end{array} \neq \begin{array}{|c c c|}\hline 4 & 5 & 3 \\ \hline 2 & 1 \\ \cline{1-2}\end{array}$$

1.3.3 The modules M^λ and S^λ

Before defining M^λ, we must first linearly extend S_n to the group ring $\mathbb{C}[S_n]$, the

formal linear combinations of permutations with coefficents in $\mathbb{C}$ — $\sum x_\sigma \sigma$. Multiplication in the group ring is determined by composition. Note that we can restrict a $\mathbb{C}[S_n]$-module to an S_n-module by restricting $\mathbb{C}[S_n]$ to the set

$$\{1_{\mathbb{C}}\sigma | \sigma \in S_n\}$$

which is isomorphic to S_n, so we still obtain our desired S_n-module and can construct any desired matrix representations of S_n.

Definition 1.3.7 ([Ful, Pg. 86])**.** We let M^λ denote the complex vector space with basis the set of tabloids $\{T\}$ for a given partition λ size n.

Since S_n acts on the set of tabloids, it acts on M^λ, and we can linearly extend this to an action of $\mathbb{C}[S_n]$ on M^λ; namely

$$(\sum x_\sigma \sigma) \cdot (\sum x_T \{T\}) = \sum \sum x_\sigma x_T \{\sigma \cdot T\}.$$

Therefore M^λ is a $\mathbb{C}[S_n]$-module.

We now need some special elements to construct our desired submodule of M^λ:

Definition 1.3.8 ([Ful, Pg. 86])**.** Given a tableaux T, we define the element $b_T \in \mathbb{C}[S_n]$:

$$b_T = \sum_{q \in C(T)} \operatorname{sgn}(q) q.$$

This is a *Young symmetrizer*; there are two others we have left undefined as they are used in the symmetric (column-wise) definition of tabloids outside of our scope.

Definition 1.3.9 ([Ful, Pg. 86])**.** For each tableaux (not tabloid!) T of shape λ there is an element $v_T \in M_\lambda$ defined by the formula

$$v_T = b_T \cdot \{T\} = \sum \operatorname{sgn}(q) \{q \cdot T\}.$$

Note that changing T might not change the tabloid $\{T\}$ but could change b_T resulting in a different element in M^λ.

We can now define the subspace S^λ:

Definition 1.3.10 ([Ful, Pg. 87])**.** The *Specht module*, denoted S^λ, is the subspace of M^λ spanned by the elements v_T, as T varies over all tableaux of λ.

To prove that S^λ is actually a module, we need to show that it is closed under the action of $\mathbb{C}[S_n]$; namely, we will show that $\sigma \cdot v_T = v_{\sigma \cdot T}$ for all tableaux T and $\sigma \in S_n$.

Proof. Recall that $C(\sigma \cdot T) = \sigma \cdot C(T) \cdot \sigma^{-1}$; we first observe that

$$\begin{aligned} \sigma \cdot v_T &= \sigma \cdot b_T \cdot \{T\} \\ &= \sigma \cdot \sum_{q \in C(T)} \operatorname{sgn}(q)\{q \cdot T\} \\ &= \sum_{q \in C(T)} \operatorname{sgn}(q)\{\sigma \cdot q \cdot T\}. \end{aligned}$$

On the other hand,

$$\begin{aligned} v_{\sigma \cdot T} &= b_{\sigma \cdot T}\{\sigma \cdot T\} \\ &= \sum_{q \in C(\sigma \cdot T)} \operatorname{sgn}(q)\{q \cdot \sigma \cdot T\} \\ &= \sum_{q \in \sigma \cdot C(T) \cdot \sigma^{-1}} \operatorname{sgn}(q)\{q \cdot \sigma \cdot T\} \\ &= \sum_{q \in C(T)} \operatorname{sgn}(\sigma q \sigma^{-1})\{\sigma q \sigma^{-1} \cdot \sigma \cdot T\} \\ &= \sum_{q \in C(T)} \operatorname{sgn}(\sigma q \sigma^{-1})\{\sigma \cdot q \cdot T\} \end{aligned}$$

We cite the abstact alegebra fact that conjugation doesn't change the sign of a permutation, so these two expressions are equal. □

Because $\sigma \cdot v_T = v_{\sigma \cdot T} \in S^\lambda$, S^λ is closed under S_n. Furthermore S^λ is closed under $\mathbb{C}[S_n]$, as we can now calculate

$$(\sum x_\sigma \sigma) \cdot v_T = \sum x_\sigma v_{\sigma \cdot T} \in S^\lambda.$$

Because S^λ is a subspace of M^λ, we have thus proven that

Theorem 1.3.11. *S^λ is a $\mathbb{C}[S_n]$ submodule of M^λ.*

In fact, we know that $S^\lambda = \mathbb{C}[S_n] \cdot v_T$ for *any given* tableaux T, because we can

create any desired element in S^λ like so:

$$\sum_{\{T'\}\in A\subseteq \text{ set of tabloids}} x_{T'} v_{T'} = (\sum x_{T'}\sigma_{T'}) \cdot v_T,$$

where $\sigma_{T'} \cdot v_T = v_{\sigma_{T'} \cdot T} = v_{T'}$.

1.3.4 Irreducibility and completeness of the S^λ modules

We now need to show that the modules S^λ have our desired properties: every S^λ is irreducible, no two S^λ are isomorphic, and any irreducible representation is isomorphic to some S^λ. Once we do this we will have shown that the set of modules S^λ over partitions of a given size n are in bijection with the irreducible representations of S_n (up to isomorphism)!

The idea of our proof will be to show that each S^λ is indecomposable (contains no nontrivial submodules) and distinct by putting a linear order on the tableaux of any shape and of size n. Matchske's theorem tells us that indecomposibility is equivalent to irreducibility, and we will use some properties of the ordering to show that each S^λ is disjoint to $S^{\lambda'}$ when $\lambda < \lambda'$ and $\lambda \neq \lambda'$.

We begin by defining the lexicographical and dominance orderings:

Definition 1.3.12 ([Ful, pg. 36])**.** The *lexicographic* ordering on partitions of size n, denoted $\lambda \leq \lambda'$, means that for the first i for which $\lambda_i \neq \lambda'_i$ (if any), has $\lambda_i < \lambda'_i$ (in the standard ordering on integers). It is a linear order.

Definition 1.3.13. The *dominance* ordering on partitions of size n, denoted $\lambda \trianglelefteq \lambda'$ or "λ' dominates λ", means that $\sum_{1\leq j\leq i} \lambda_j \leq \sum_{1\leq j\leq i} \lambda'_j$ for all $1 \leq i \leq \infty$.

The intuition here is that parititions with a few long rows dominate partitions with many short rows; note, however, that this is not a linear order. For example, we see for the partitions $\lambda = (4,1,1,1) \vdash 7$ and $\lambda' = (3,3,1) \vdash 7$ that

$$4 \not\leq 3, \text{ so } \lambda \not\trianglelefteq \lambda'$$
$$3 \leq 4,\ 3+3 \leq 4+1 \text{ so } \lambda' \not\trianglelefteq \lambda$$

so λ and λ' are incomparable.

We can now state the following lemma:

Lemma 1.3.14 ([Ful, Lemma 7.1]). *Let T and T' be tableaux of shape λ and λ' respectively, each of size n. Note that they can have different shape! Assume that λ does not strictly dominate λ'. Then exactly one of the following occurs:*

1. *There are two distinct integers that occur in the same row of T' and the same column of T.*

2. *$\lambda = \lambda'$, and there is some p' in $R(T)$ and some q in $C(T)$ such that $p' \cdot T' = q \cdot T$.*

Proof. Suppose 1 is false. The entries of the first row of T' must occur in different columns of T, so there is a $q_1 \in C(T)$ so that these entires occur in the first row of $q \cdot T$.

The entries of the second row of T' occur in different columns in T, and so they also occur in different columns of $q_1 \cdot T$, so there is a $q_2 \in C(q_1 \cdot T) = C(T)$ that:

1. Doesn't move entries in $q_1 \cdot T$ that are also in the first row of T'.

2. Moves entries in the second row of T' that are in $q_1 \cdot T$ into the second row of $q_1 \cdot T$.

We can repeat this process to obtain $q_1, \ldots, q_k$ such that the entries in the first k rows of T' occur in the first k rows of $q_k \cdot q_{k-1} \cdot \cdots \cdot q_1 \cdot T$. The actions of $q_1, \ldots, q_k$ don't change the shape of T, so we can deduce that

$$\sum_{1 \leq j \leq k} \lambda'_j \leq \sum_{1 \leq j \leq k} \lambda_j$$

because each row of λ must have at least enough boxes to contain all the entries of the corresponding row of λ'. This holds for all k, so by definition λ dominates λ'.

We assumed that λ doesn't strictly dominate λ', so the only possibility is that $\lambda = \lambda'$. We can thus take k to be the number of rows in λ and let $q = q_k \cdots q_1$ so that $q \cdot T$ and T' have the same entries in each row. We can now conclude that there is some $p' \in R(T)$ such that $p' \cdot T' = q \cdot T$. □

We now define our needed linear ordering; note that this ordering is on all tableaux of size n, not just partitions!

Definition 1.3.15 ([Ful, pg. 84-85]). We denote the linear ordering $T < T'$ on tableaux to mean either:

1. The shape of T' is larger than the shape of T in the lexicographical order.

2. T and T' have the same shape, and the largest entry that is in a different box in the two numberings occurs earlier in the column word of T' than in the column word of T.

   The column word is obtained by listing the entries of each column from bottom to top, reading columns from left to right.

Example 1.3.16. This ordering puts the standard tableux of shape $(3, 2)$ in the following order:

$$\begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4&5\\\cline{1-2}\end{array} > \begin{array}{|c|c|c|}\hline 1&2&4\\\hline 3&5\\\cline{1-2}\end{array} > \begin{array}{|c|c|c|}\hline 1&3&4\\\hline 2&5\\\cline{1-2}\end{array} > \begin{array}{|c|c|c|}\hline 1&2&5\\\hline 3&4\\\cline{1-2}\end{array} > \begin{array}{|c|c|c|}\hline 1&3&5\\\hline 2&4\\\cline{1-2}\end{array}.$$

We demonstrate the first inequality by observing that 4 is the largest entry in a different box between the two SYT, and that the first SYT has column word 41523 while the second has column word 31524. We see that 4 occurs earlier in the column word of the first SYT, so the first SYT is "larger" in this ordering.

An important property of this ordering for SYT T and any $p \in R(T), q \in C(T)$ is that

$$p \cdot T < T \text{ and } q \cdot T > T.$$

The symbol "$<$" does not denote a strict poset relation, so it could be that $p \cdot T = T$.

This is true because the largest element moved by a row permutation must be moved left, pushing it closer to the front in the column word, while the largest element moved by a column permutation must be moved up, pushing it further back in the column word.

Example 1.3.17. In the SYT below, any nontrivial row permutation will swap at least two elements. Because every element to the right of a given entry in a row is larger in a SYT, this swap will move the larger element to the left. Analogously, elements lower in a column are larger, so a column permutation will move the larger element up.

$$(15) \cdot \begin{array}{|c|c|c|c|c|}\hline 1&2&3&4&5\\\hline\end{array} = \begin{array}{|c|c|c|c|c|}\hline 5&2&3&4&1\\\hline\end{array}$$

We need the following lemmas for the next 2 proofs:

Lemma 1.3.18. *For all $q' \in C(T)$,*

$$q' \cdot b_T = \operatorname{sgn}(q') b_T.$$

Proof. First we compute

$$q' \cdot b_T = q' \cdot \sum_{q \in C(T)} \operatorname{sgn}(q)\{q \cdot T\} = \sum_{q \in C(T)} \operatorname{sgn}(q)\{q' \cdot q \cdot T\}.$$

We know from abstract algebra that $q'C(T) = C(T)$ because $q' \in C(T) \leq S_n$; the function of applying q' to every element of $C(T)$ is a bijection. Therefore the composition of q' with every element of $C(T)$ will only reorder the addends in the sum. If q' is odd, it will map each permutation to a permutation with opposite sign; if it is even, it will maintain the parity of each permutation. This is equivalent to multiplying each permutation by the sign of q', so

$$\sum_{q \in C(T)} \operatorname{sgn}(q)\{q' \cdot q \cdot T\} = \sum_{q \in C(T)} \operatorname{sgn}(q)\operatorname{sgn}(q')\{q \cdot T\} = \operatorname{sgn}(q') \sum_{q \in C(T)} \operatorname{sgn}(q)\{q \cdot T\} = \operatorname{sgn}(q') b_T.$$

□

Lemma 1.3.19.

$$b_T \cdot b_T = |C(T)| \cdot b_T,$$

where $\cdot$ here is multiplication in the group ring.

Proof. We linearly extend the result we just proved:

$$b_T \cdot b_T = \Big(\sum_{q \in C(T)} \operatorname{sgn}(q) q\Big) \cdot b_T = |C(T)| \operatorname{sgn}(q)\operatorname{sgn}(q) \cdot b_T = |C(T)| \cdot b_T.$$

□

We now state and prove another lemma which applies our previous lemma to the M^λ module:

Lemma 1.3.20 ([Ful, Lemma 7.2])**.** *Let T and T' be numberings of shapes λ and λ' respectively, and assume that λ does not strictly dominate λ'.*

1. *If there is a pair of integers in the same row of T' and the same column of T, then $b_T \cdot \{T'\} = 0$.*
2. *If there is no such pair, then $b_T \cdot \{T'\} = \pm v_T$.*

Proof. If there is such a pair of integers, let $t \in S_n$ be the transposition that swaps them. Then $b_T \cdot t = -b_T$, since t is in the column group of T, and transpositions have

odd sign. On the other hand $t \cdot \{T'\} = \{T'\}$ by the definition of a tabloid, because t is in the row group of T'. Therefore

$$b_T \cdot \{T'\} = b_T \cdot (t \cdot \{T'\}) = (b_T \cdot t) \cdot \{T'\} = -b_T \cdot \{T'\}$$

so $b_T \cdot \{T'\} = 0$.

If there is no such pair of integers, then let p' and q be as in the second case of our first lemma. Then

$$\begin{aligned} b_T \cdot \{T'\} &= b_T\{p' \cdot T'\} = b_T \cdot \{q \cdot T\} \\ &= b_T \cdot q \cdot \{T\} = \operatorname{sgn}(q) b_T \cdot \{T\} = \operatorname{sgn}(q) v_T. \end{aligned}$$

□

Finally we have the tools to prove our main theorem!

Theorem 1.3.21 ([Ful, Pg. 87-88])**.** *For each partition λ of n, S^λ is an irreducible representation of S_n. Every irreducible representation of S_n is isomorphic to exactly one S_n.*

Proof. First, we note that no v_T is 0 by definition, so the modules S^λ are all nonzero (nontrivial subspaces of M^λ). We wish to prove the following statements, for a given tableaux T of λ:

$$b_T \cdot M^\lambda = b_T \cdot S^\lambda = \mathbb{C} \cdot v_T \neq \{0\}. \tag{1.3.22}$$

$$b_T \cdot M^{\lambda'} = b_T \cdot S^{\lambda'} = \{0\} \text{ if } \lambda < \lambda' \text{ and } \lambda \neq \lambda', \tag{1.3.23}$$

where $\{0\}$ is the trivial zero subspace. We begin with the first equation.

The first equality follows as such, using our result that $b_T \cdot b_T = |C(T)| \cdot b_T$:

$$\begin{aligned} b_T \cdot b_T &= |C(T)| \cdot b_T \iff \\ \frac{1}{|C(T)|} \cdot b_T \cdot b_T &= b_T; \\ b_T \cdot M^\lambda &= \frac{1}{|C(T)|} \cdot b_T \cdot b_T \cdot M^\lambda \\ &= \frac{1}{|C(T)|} \cdot b_T S^\lambda \\ &= b_T \cdot \frac{1}{|C(T)|} \cdot S^\lambda \\ &= b_T \cdot S^\lambda \end{aligned}$$

where we can commute $\frac{1}{|C(T)|}$ because it is only a scalar; furthermore $\frac{1}{|C(T)|} \cdot S^\lambda = S^\lambda$ because vector spaces are invariant under scaling. The second equality follows because we know that for any T, T' tableaux of λ, we know by Lemma 1.3.14 there are not two distinct integers that appear in the same row of T' and in the same column of T, so by Lemma 1.3.20 $b_T \cdot \{T'\} = \pm v_T$. This means that $b_t \cdot M^\lambda = \mathbb{C} \cdot v_T$, because every element in M^λ is mapped to a scaling of v_T, and any we can obtain any scaling of v_T via $b_T \cdot x\{T\} = xv_T \in \mathbb{C} \cdot v_T$.

For the second equation, the first equality follows from our argument regarding the second equation. The second equality follows because by assumption we are in case 1 of Lemma 1.3.14 and therefore case 1 of Lemma 1.3.20, so $b_T \cdot \{T'\} = 0$ for all $\{T'\}M^\lambda$, and so $b_T \cdot M^\lambda = 0$.

Now by Matchske's theorem we know that $S^\lambda = W_1 \oplus W_2 \oplus \cdots \oplus W_k$ for irreducible submodules W_j. We see that

$$\mathbb{C} \cdot v_T = b_T \cdot S^\lambda = b_T \cdot W_1 \oplus b_T \cdot W_2 \cdots \oplus b_T \cdot W_k,$$

so one of the modules W_j must contain v_T. If some W_i contains v_T, then by the definition of a submodule $W_i = \mathbb{C}[S_n] \cdot v_T = S^\lambda$, so the other submodules must be the zero submodule and so S^λ is irreducible.

Furthermore we prove that $S^\lambda \ncong S^{\lambda'}$ for any $\lambda < \lambda'$ and $\lambda \neq \lambda'$. We assume there exists a module isomorphism Θ between S^λ and $S^{\lambda'}$. By the definition of a module isomorphism, $\Theta(b_T \cdot S^{\lambda'}) = b_T \cdot \Theta(S^{\lambda'})$, but we just showed that

$$\Theta(b_T \cdot S^{\lambda'}) = \Theta(0) = 0 \neq b_T \cdot S^\lambda = b_T \cdot \Theta(S^{\lambda'})$$

which is a contradiction. Therefore $S^\lambda \ncong S^{\lambda'}$, and because $<$ is a linear ordering, we can conclude that no two distinct S^λ are isomorphic.

Finally we cite without proof [Sag, Proposition 1.10.1], specifically the result that the number of irreducible modules/representations equals the number of conjugacy classes of the group. We have previously discussed how the conjugacy classes of S_n are in bijection with cycle types of S_n, which in turn are in bijection with partitions of n. Because there is one Specht module S^λ for each partition λ of size n, we can conclude that there are exactly as many S^λ as there are irreducible representations of S_n. That is, the set of modules S^λ is all of the irreducible modules/representations of S_n up to isomorphism. $\square$

1.3.5 A basis for S^λ

With the main result aside, we now prove a proposition that first requires a combinatorial result.

Proposition 1.3.24 ([Ful, Pg. 88]). *The elements v_T, as T varies over the* standard *tableaux of λ, form a basis for S^λ.*

define standard tableaux

Proof. The element v_T is a linear combination of $\{T\}$, with coefficent 1 (as the trivial permutation has even sign), and elements $\{q \cdot T\}$, for $q \in C(T)$, with coefficents ± 1. Recall that when T is a SYT, $q \cdot T < T$, and furthermore this relation is strict when q is nontrivial. To find solutions to the equation $\sum_{T \in \text{ SYT of } \lambda} x_T v_T = 0$, we can look at the largest v_T with nonzero coefficent x_T. We know that the $\{T\}$ component cannot be canceled out by some other $v_{T'}$, because it must be that $\{T\} \neq \{T'\}$ in order for the relation $v_T > v_{T'}$ to be strict, and furthermore $\{T\} \neq \{q \cdot T'\}$ because we know $\{T\} > \{T'\} > \{q \cdot T'\}$ and at least the first relation is strict. Thus $\{T\}$ cannot be canceled out by some other term, so it must be that $x_T = 0$; but then all $x_T = 0$ because if any x_T is nonzero there will be some largest nonzero v_T. We can thus conclude that the elements $v_T \in S^\lambda$ as T varies over the SYT are linearly independent.

To show that these elements span S^λ, we again cite [Sag, Proposition 1.10.1], specifically the result that

$$\sum_i (\dim V_i)^2 = |G|$$

where the V_i are a complete set of irreducible G-modules. In our case this means that

$$\sum_\lambda (\dim S^\lambda)^2 = n!$$

We proved in class that

$$\sum_\lambda (f^\lambda)^2 = n!$$

I have to do this proof myself now

where f^λ is the number of SYT of the partition λ of size n. We can thus conclude

$$n! = \sum_\lambda (\dim S^\lambda)^2 = \sum_\lambda (f^\lambda)^2 = n!.$$

It follows that $\dim(S^\lambda) = f^\lambda$ for all λ, and because f^λ counts the number of SYT of shape λ this means that the elements v_T as T varies over SYT must span S^λ. □

1.4 Oracle Formalism

We are now equipped to state our oracle problem and key theorem formally.

Definition 1.4.1 ([CP, Section 2]). A *classical oracle problem* is a tuple (Y, Ω, π, f) where

1. Y is a set of hidden information.
2. Ω is a set of inputs algorithms can query.
3. π is a function $\pi : Y \to \mathrm{Sym}(\Omega)$, where $\mathrm{Sym}(\Omega)$ is the group of permutations of Ω.
4. And $f : Y \to X$ is the function to learn; it is known to the algorithm.

A classical computer has access to $\pi(y)$ for some unknown $y \in Y$ by spending one query to learn $\pi(y) \cdot \omega$. The goal is to determine $f(y)$. The average case case success probability is the probability of correctly outputting $f(y)$ assuming y is sampled uniformly from Y.

Definition 1.4.2 ([CP, Section 2]). A *quantum oracle problem* is a tuple (Y, V, π, f) where

1. Y is a set of hidden information.
2. V is a Hilbert space appearing as a register in our quantum circuit.
3. π is a function $\pi : Y \to U(V)$, where $U(V)$ is the set of unitary operators of V.
4. And $f : Y \to X$ is the function to learn.

A quantum computer spends one query to input a state $|\psi\rangle \in V$ to $\pi(y)$ to acquire the state $\pi(y)|\psi\rangle$. Any classical oracle problem (Y, ω, π, f) determines a quantum oracle problem via linearization: oracles will act on the Hilbert space $\mathbb{C}\Omega$ by permutation matrices. Henceforth when we provide an instance of a classical oracle problem we are also providing an instance of a quantum oracle problem.

doesn't Omega need to be an abelian group?

Definition 1.4.3 ([CP, Section 2]). A *symmetric oracle problem* is a classical/quantum oracle problem where

1. we require that the hidden information Y be a group G.
2. Ω or V are as they were before.

3. π is as before, but now also is a homomorphism to the group $\mathrm{Sym}(\Omega)$ in the classical case and to the group $U(V)$ in the quantum case.

4. And f is as it was before.

When $\pi : G \to U(V)$ is a homorphism, it is a representation as we have defined before; it is therefore natural to regard V as a (left) $\mathbb{C}G$-module where $\mathbb{C}G$ is the group algebra of G (spanned by the orthonormal basis $\{g|g \in G\}$ where we leave out kets to indicate their use as an action), due to the relationship between representations and modules.

Definition 1.4.4 ([CP, Section 2])**.** A *coset identification problem* is a symmetric oracle problem where the function to be learned $f : G \to X$ is constant on left cosets of a subgroup $H \leq G$ and distinct on distinct cosets. We also assume f is surjective. The typical example is when $X = \{gH|g \in G\}$ is the set of left cosets of H and $f(g) = gH$. We cite without proof the fact that worst case and average case success probabilities are equal in the coset identification problem.

Definition 1.4.5 ([CP, Section 2])**.** The *exact* query complexity of a learning problem is the minimum number of queries needed by an algorithm to compute $f(y)$ with zero probability of error. The *bounded* error query complexity is the minimum number of queries needed by an algorithm to compute $f(y)$ with probability $\geq 2/3$. The bounded error query complexity is often studied for a family of problems growing with a parameter n and so changing the constant $2/3$ above to any number strictly greater than $1/2$ will only change the query complexity by a constant factor mostly ignored in asymptotic analysis.

Definition 1.4.6 ([CP, Section 4])**.** *Symmetric oracle discrimination* is a coset identification problem a la the typical example where $X = \{gH|g \in G\}$, $f(g) = gH$, and $H = \{e\}$, so $f(g) = ge = g$. This means an element g is hidden and the goal is to find g. We have fixed f, so such a problem is determined by a choice of finite group G, a G-set Ω or a vector space $\mathbb{C}\Omega$, and an action $G \circlearrowright \Omega$ or a (finite dimensional) unitary representation $\pi : G \to U(V)$.

Our problem is thus two instances of symmetric oracle discrimination where, in the first case,

1. $G = S_n$. and A_n

2. Ω is the set of k elements subsets of $[n] = \{1, 2, \cdots, n\}$.

3. $S_n \circlearrowright \Omega$ is the natural action of S_n:

$$\sigma \circlearrowright \{\omega_1, \omega_2, \cdots, \omega_k\} \mapsto \{\sigma(\omega_1), \sigma(\omega_2), \cdots, \sigma(\omega_k)\}.$$

And in the second case,

and A_n

1. $G = S_n$.

2. Ω is the set of regular partitions of $[n]$ into b parts of size a, so $ab = n$.

3. $S_n \circlearrowright \Omega$ is the natural action of S_n:

$$\sigma \circlearrowright \{\{\omega_1, \omega_2, \cdots, \omega_a\}, \{\omega_{a+1}, \omega_{a+2}, \cdots, \omega_{2a}\}, \cdots, \{\omega_{(a-1)b+1}, \omega_{(a-1)b+2}, \cdots, \omega_{ab}\}\} \mapsto$$
$$\{\{\sigma(\omega_1), \sigma(\omega_2), \cdots, \sigma(\omega_a)\}, \{\sigma(\omega_{a+1}), \sigma(\omega_{a+2}), \cdots, \sigma(\omega_{2a})\}, \cdots,$$
$$\{\sigma(\omega_{(a-1)b+1}), \sigma(\omega_{(a-1)b+2}), \cdots, \sigma(\omega_{ab})\}\}.$$

Finally, we introduce the theorem which our results rely upon.

Theorem 1.4.7. *Suppose G is a finite group and $\pi : G \to U(V)$ a unitary representation of G. Then an optimal t-query algorithm to solve symmetric oracle discrimination succeeds with probability*

$$P_{opt} = \frac{d_{V^{\otimes t}}}{|G|}$$

where

$$d_{V^{\otimes t}} = \sum_{\chi \in I(V^{\otimes t})} \chi(e)^2.$$

Note that $I(V^{\otimes t})$ is the set of irreducible constituent characters of the character $V^{\otimes t}$.

Proof Outline 1.4.8. In a paper seperate to the one we cite, Copeland and Pommershiem proved a similar theorem for a single query algorithm, where only the irreducible consituent characters of V were considered. In [CP], Copeland and Pommershiem prove that nonadaptive (all oracle queries made in parallel) quantum algorithms have equivalent strength to adaptive (arbitary unitary transformations allowed between queries) algorithms. Furthermore, they prove that a t-query nonadaptive algorithm has equivalent success probability to a a single query algorithm which takes the tensor product of the representation $\pi : G \to U(V)$ with itself multiple times to the representation $\pi^{\otimes t} : G \to U(V^{\otimes t})$. We can thus substitute $\pi^{\otimes t} : G \to U(V^{\otimes t})$ in for the single query theorem to find the optimal t-query success chance.

This theorem tells us that we can determine the query complexity of a symmetric oracle discrimination problem just by understanding how the consituent irreducibles of the representation $\pi : G \to U(V)$ change as n grows, and how the repeated tensor product of these constituents introduces new irreducible representations. To make our work computationally possible we will use the characters of these irreducible representations to do our analysis rather than the representations themselves.

Chapter 2

Methods

We approached this problem by analyzing the query complexity and the constituent irreducibles of each action for small values of n. This was done via computation in the GAP (Group, Algorithms, and Programming) language, which offers native support for group and representation theory calculations. Computations using representations themselves are slow and space-intensive, so we study the representations we are interested in via their characters. Each action is created using built-in objects, then used to create a permutation character corresponding to the module $\mathbb{C}\Omega$, where Ω is the G-set of the action. With this character we can compute the query complexity and bounded query complexity with a simple algorithm, offered here in pseudocode:

Require: *permutationCharacter* is the permutation character of $G \circlearrowright \Omega$

procedure QUERYCOMPLEXITY(*permutationCharacter*, G)

- $queries \leftarrow 0$
- $boundedQueries \leftarrow 0$
- ▷ *tensorCharacter stores* $V^{\otimes t}$, *while permutationCharcter stores* V. ◁
- $tensorCharacter \leftarrow permutationCharacter$
- **repeat**
  - $queries \leftarrow queries + 1$
  - $constituents \leftarrow$ set of constituent characters of $tensorCharacter$
  - $probabilityOfSuccess \leftarrow \sum_{\chi \in constituents} \chi(e)^2/|G|$
  - **if** $probabiltyOfSuccess \geq \frac{2}{3}$ and $boundedQueries = 0$ **then**
    - $boundedQueries \leftarrow queries$
  - $tensorCharacter \leftarrow tensorCharacter \otimes permutationCharacter$
- **until** $constituents =$ set of irreducible characters of G

In addition, GAP offers a special character table for the symmetric group which uses the *Murnaghan-Nakayama Rule* to recursively associate a partition with each irreducible character of S_n. Using this table we were able to store the corresponding partitions of the constituent characters present after each query, which we then visualized in Python. Both these partitions and the behavior of the query complexity as n increased helped us make conjectures regarding the general query complexity of these actions.

Chapter 3

Results

Chapter 4

The First

This is the first page of the first chapter. You may delete the contents of this chapter so you can add your own text; it's just here to show you some examples.

4.1 References, Labels, Custom Commands and Footnotes

It is easy to refer to anything within your document using the `label` and `ref` tags. Labels must be unique and shouldn't use any odd characters; generally sticking to letters and numbers (no spaces) should be fine. Put the label on whatever you want to refer to, and put the reference where you want the reference. LaTeX will keep track of the chapter, section, and figure or table numbers for you.

4.1.1 References and Labels

Sometimes you'd like to refer to a table or figure, e.g. you can see in Figure 6.2 that you can rotate figures . Start by labeling your figure or table with the label command (`\label{labelvariable}`) below the caption (see the chapter on graphics and tables for examples). Then when you would like to refer to the table or figure, use the ref command (`\ref{labelvariable}`). Make sure your label variables are unique; you can't have two elements named "default." Also, since the reference command only puts the figure or table number, you will have to put "Table" or "Figure" as appropriate, as seen in the following examples:

As I showed in Table 6.1 many factors can be assumed to follow from inheritance. Also see the Figure 6.1 for an illustration.

4.1.2 Custom Commands

Are you sick of writing the same complex equation or phrase over and over?

The custom commands should be placed in the preamble, or at least prior to the first usage of the command. The structure of the `\newcommand` consists of the name of the new command in curly braces, the number of arguments to be made in square brackets and then, inside a new set of curly braces, the command(s) that make up the new command. The whole thing is sandwiched inside a larger set of curly braces.

In other words, if you want to make a shorthand for H_2SO_4, which doesn't include an argument, you would write: `\newcommand{\hydro}{H$_2$SO$_4$}` and then when you needed to use the command you would type `\hydro`. (sans verb and the equals sign brackets, if you're looking at the .tex version). For example: H_2SO_4

4.1.3 Footnotes and Endnotes

You might want to footnote something.[1] Be sure to leave no spaces between the word immediately preceding the footnote command and the command itself. The footnote will be in a smaller font and placed appropriately. Endnotes work in much the same way. More information can be found about both on the CUS site.

4.2 Bibliographies

Of course you will need to cite things, and you will probably accumulate an armful of sources. This is why BibTeX was created. For more information about BibTeX and bibliographies, see our CUS site (`web.reed.edu/cis/help/latex/index.html`). There are three pages on this topic: *bibtex* (which talks about using BibTeX, at `/latex/bibtex.html`), *bibtexstyles* (about how to find and use the bibliography style that best suits your needs, at `/latex/bibtexstyles.html`) and *bibman* (which covers how to make and maintain a bibliography by hand, without BibTeX, at at `/latex/bibman.html`). The last page will not be useful unless you have only a few sources. There used to be APA stuff here, but we don't need it since I've fixed this with my apa-good natbib style file.

[1] footnote text

4.2.1 Tips for Bibliographies

1. Like with thesis formatting, the sooner you start compiling your bibliography for something as large as thesis, the better. Typing in source after source is mind-numbing enough; do you really want to do it for hours on end in late April? Think of it as procrastination.

2. The cite key (a citation's label) needs to be unique from the other entries.

3. When you have more than one author or editor, you need to separate each author's name by the word "and" e.g.
   `Author = {Noble, Sam and Youngberg, Jessica},`.

4. Bibliographies made using BibTeX (whether manually or using a manager) accept LaTeX markup, so you can italicize and add symbols as necessary.

5. To force capitalization in an article title or where all lowercase is generally used, bracket the capital letter in curly braces.

6. You can add a Reed Thesis citation option. The best way to do this is to use the phdthesis type of citation, and use the optional "type" field to enter "Reed thesis" or "Undergraduate thesis". Here's a test of Chicago, showing the second cite in a row being different. Also the second time not in a row should be different. Of course in other styles they'll all look the same.

4.3 Anything else?

If you'd like to see examples of other things in this template, please contact CUS (email cus@reed.edu) with your suggestions. We love to see people using LaTeX for their theses, and are happy to help.

Chapter 5

Mathematics and Science

5.1 Math

TEX is the best way to typeset mathematics. Donald Knuth designed TEX when he got frustrated at how long it was taking the typesetters to finish his book, which contained a lot of mathematics.

If you are doing a thesis that will involve lots of math, you will want to read the following section which has been commented out. If you're not going to use math, skip over this next big red section. (It's red in the .tex file but does not show up in the .pdf.)

Chapter 6

Tables and Graphics

6.1 Tables

The following section contains examples of tables, most of which have been commented out for brevity. (They will show up in the .tex document in red, but not at all in the .pdf). For more help in constructing a table (or anything else in this document), please see the LaTeX pages on the CUS site.

Table 6.1: Correlation of Inheritance Factors between Parents and Child

Factors	Correlation between Parents & Child	Inherited
Education	-0.49	Yes
Socio-Economic Status	0.28	Slight
Income	0.08	No
Family Size	0.19	Slight
Occupational Prestige	0.21	Slight

If you want to make a table that is longer than a page, you will want to use the longtable environment. Uncomment the table below to see an example, or see our online documentation.

6.2 Figures

If your thesis has a lot of figures, LaTeX might behave better for you than that other word processor. One thing that may be annoying is the way it handles "floats" like tables and figures. LaTeX will try to find the best place to put your object based on the text around it and until you're really, truly done writing you should just leave it where it lies. There are some optional arguments to the figure and table environments to specify where you want it to appear; see the comments in the first figure.

If you need a graphic or tabular material to be part of the text, you can just put it inline. If you need it to appear in the list of figures or tables, it should be placed in the floating environment.

To get a figure from StatView, JMP, SPSS or other statistics program into a figure, you can print to pdf or save the image as a jpg or png. Precisely how you will do this depends on the program: you may need to copy-paste figures into Photoshop or other graphic program, then save in the appropriate format.

Below we have put a few examples of figures. For more help using graphics and the float environment, see our online documentation.

And this is how you add a figure with a graphic:

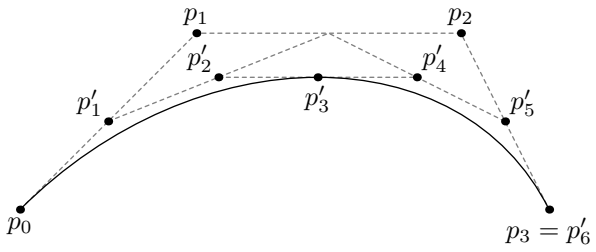


Figure 6.1: A Figure

6.3 More Figure Stuff

You can also scale and rotate figures.

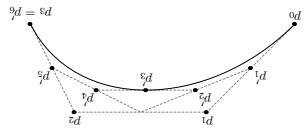


Figure 6.2: A Smaller Figure, Flipped Upside Down

6.4 Even More Figure Stuff

With some clever work you can crop a figure, which is handy if (for instance) your EPS or PDF is a little graphic on a whole sheet of paper. The viewport arguments are the lower-left and upper-right coordinates for the area you want to crop.

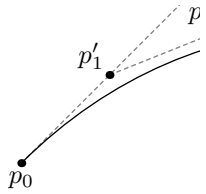


Figure 6.3: A Cropped Figure

6.4.1 Common Modifications

The following figure features the more popular changes thesis students want to their figures. This information is also on the web at `web.reed.edu/cis/help/latex/graphics.html`.

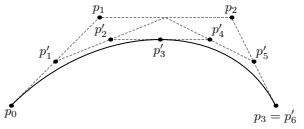


Figure 6.4: Subdivision of arc segments. You can see that $p_3 = p'_6$.

Conclusion

Here's a conclusion, demonstrating the use of all that manual incrementing and table of contents adding that has to happen if you use the starred form of the chapter command. The deal is, the chapter command in LaTeX does a lot of things: it increments the chapter counter, it resets the section counter to zero, it puts the name of the chapter into the table of contents and the running headers, and probably some other stuff.

So, if you remove all that stuff because you don't like it to say "Chapter 4: Conclusion", then you have to manually add all the things LaTeX would normally do for you. Maybe someday we'll write a new chapter macro that doesn't add "Chapter X" to the beginning of every chapter title.

4.1 More info

And here's some other random info: the first paragraph after a chapter title or section head *shouldn't be* indented, because indents are to tell the reader that you're starting a new paragraph. Since that's obvious after a chapter or section title, proper typesetting doesn't add an indent there.

References

[BMMN] Meenaxi Bhattacharjee, Dugald Macpherson, Rögnvaldur G. Möller, and Peter M. Neumann. Wreath products. In *Notes on Infinite Permutation Groups*, pages 67–76. Springer.

[CP] Daniel Copeland and Jamie Pommersheim. Quantum query complexity of symmetric oracle problems.

[fam] prefix=del useprefix=false family=Valle, given=Coen. A character theoretic formula for base size.

[Ful] William Fulton. *Young Tableaux: With Applications to Representation Theory and Geometry*. Cambridge University Press.

[JL] Gordon James and Martin Liebeck. *Representations and Characters of Groups*. Cambridge University Press, 2 edition.

[NC] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press.

[Sag] Bruce E. Sagan. *The Symmetric Group*, volume 203 of *Graduate Texts in Mathematics*. Springer.

[Zha] Mark Zhandry. Quantum Oracle Classification - The Case of Group Structure.