My Final College Paper

A Thesis
Presented to
The Division of Mathematics and Natural Sciences
Reed College

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

Your R. Name

May 200x

Approved for the Division
(Mathematics)

_____

Advisor F. Name

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

Introduction **1**
0.1 Forward 1
0.2 Graph Theory in Systems Biology 1
0.2.1 Graphs and Graph Algorithms 2
0.3 Hypergraphs 3
0.4 A Crash Course in P Versus NP 3
0.5 Parameterized Complexity: Practical Solutions to Theoretically Impossible Problems 8
0.6 Why use it? 8
0.7 Who should use it? 8

Chapter 1: The First **9**
1.1 References, Labels, Custom Commands and Footnotes 9
1.1.1 References and Labels 9
1.1.2 Custom Commands 9
1.1.3 Footnotes and Endnotes 10
1.2 Bibliographies 10
1.2.1 Tips for Bibliographies 10
1.3 Anything else? 11

Chapter 2: Mathematics and Science **13**
2.1 Math 13
2.2 Chemistry 101: Symbols 13
2.2.1 Typesetting reactions 14
2.2.2 Other examples of reactions 14
2.3 Physics 14
2.4 Biology 14

Chapter 3: Tables and Graphics **15**
3.1 Tables 15
3.2 Figures 17
3.3 More Figure Stuff 19
3.4 Even More Figure Stuff 19
3.4.1 Common Modifications 19

Conclusion **21**
4.1 More info 21

Appendix A: The First Appendix **23**

Appendix B: The Second Appendix, for Fun **25**

References **27**

List of Tables

3.1 Correlation of Inheritance Factors between Parents and Child 15
3.2 Chromium Hexacarbonyl Data Collected in 1998–1999 16

List of Figures

2.1 Combustion of glucose 14

3.1 A Figure 18
3.2 A Smaller Figure, Flipped Upside Down 19
3.3 A Cropped Figure 19
3.4 Subdivision of arc segments 19

Abstract

The preface pretty much says it all.

Dedication

You can have a dedication here if you wish.

Introduction

0.1 Forward

This is an interdisciplinary thesis that combines Computer Science and Biology. In this background chapter I have written sections designed to make the computer science more accessible to the biologists, sections designed to make the biology more accessible to the computer scientists, and hopefully in tandem these sections will make everything a bit more accessible to the reader who knows neither. For this reason, information presented in my background chapter may oscillate between trivially familiar and dauntingly foreign, depending on each reader's varying base of knowledge. So feel free to read up on the additional resources I present if you're not comfortable with the foundational concepts, or to skip around if you already know what I'm talking about.

I keep want
revise the or
the sections
they all sort
into each oth
not sure if it
propriate to
this here? I
it to be a sh
planation of
pository styl

0.2 Graph Theory in Systems Biology

Systems biology is a modern, holistic, modeling-focused approach to the study of life.

Biologically speaking, this thesis is primarily concerned with modeling how proteins interact. But in order to understand why protein interactions are important, it's important to have some context about how cells work. So if you're not familiar with the biological side of things, consider this a small primer. Most of what you need to know is encapsulated in a rule of thumb called the Central Dogma of Biology. My explanation of this rule will contextualize what proteins are, what they do, and why they're important. But to do that we have to start with DNA. The Central Dogma of Biology states that information in the cell flows in a stream from DNA to RNA to protein (in general)[1]. DNA, DeoxyriboNucleic Acid, is a molecule that can be arranged into long chains called polymers. The individual links of the chains, the subunits of the polymer, consist of a sugar backbone and a modular base. The base can be Adenine, Thymine, Cytosine, or Guanine (shortened A T C or G), and various cellular apparatuses can distinguish between these bases. Just like a computer uses sequences of bits in memory to encode numbers, a cell uses sequences of bases on

[1]For almost every rule in biology there is at least one exception. This is even true of the Central Dogma. For example, there are viruses who encode information in RNA and use a special enzyme to transfer that information to DNA when they infect a host. But in almost every organism, we observe that the Central Dogma holds true.

DNA polymers to encode information. In fact, it's reasonable to think about the DNA in a cell like the memory in a computer: slow-access, long-term information storage. The DNA is where the cell stores recipes (called genes) for making tools that it uses to take actions. That may sound weirdly generalized, but that's because the concept is very general. Basically everything a cell does - grow, divide, absorb energy, synthesize compounds, respond to stimuli, etc. - is encoded in its DNA. But this is not the whole story. Encoding something into DNA does not instantly make it so. Instead, the functions encoded in DNA must be executed by the cellular system in order to take effect. The next piece of the Central Dogma, RNA, is a major part of how this execution occurs.

RNA, RiboNucleic Acid, is a molecule similar to DNA. Like DNA, it can be arranged into long chains called polymers. Like DNA, each subunit in the RNA polymer consists of a sugar backbone and a modular base. Like DNA, there are four possible modular bases which are used in sequence to encode information. Indeed, RNA is quite similar to DNA, in fact their molecular structures only differ by the presence or absence of a single Oxygen atom. But the *function* of RNA in the cell is distinct. If the DNA in a cell is like the memory of a computer, then the RNA in a cell is kind of like the RAM: quick-access, short-term information storage. DNA stores recipes for making tools, but in order to actually use them, the cell first **transcribes**[2] the recipe into RNA form. To do this, the cell reads the information off of a gene in the DNA and creates a short and portable RNA polymer that encodes the same[3] information. This polymer is called a messenger[4] RNA (mRNA). The utility of using mRNA to convey information is manifold. In particular, it allows the cell to relay information to distal parts of itself while keeping the DNA sequestered in a secure location: mRNA copies of genes are free to move about the cell without the risk of damaging the master copy contained in the DNA. Additionally, by transcribing the same gene multiple times multiple mRNA copies can be made. This makes the information encoded in the gene accessible to multiple pieces of cellular apparatus at a time which greatly increases the speed of information propagation, which is important when the cell needs a widespread effect to occur.

0.2.1 Graphs and Graph Algorithms

- rigorously define what a graph is (probably include some snazzy figures to demonstrate)
- present some graph algorithms, their applications in the context of systems

[2]Transcription is an important key word that refers specifically to the synthesis of an RNA polymer from a gene in the DNA.

[3]In reality, this is sort of an oversimplification. Things like post-transcriptional modification (e.g. pre-mRNA splicing) often cause a piece of mRNA to have information content that differs from the gene from which it was transcribed. But the less biologically inclined reader need not concern themselves with this detail if their goal is to develop a cursory understanding of the Central Dogma.

[4]We make this distinction because there are other functions fulfilled by RNA in the cell, but for the purposes of this thesis a less biologically inclined reader need not know them. For a cursory working understanding of the Central Dogma, it is sufficient to focus on messenger RNA.

biology

0.3 Hypergraphs

0.4 A Crash Course in P Versus NP

Any currently discovered algorithm that we might refer to as 'efficient' is in a class called P. P stands for Polynomial time. Members of the class P can be solved in a number of timesteps that is a polynomial of the size of the input. For example, if I had a list of length n and I wanted to write that list in reverse order, an algorithm to do that might look something like this:

Algorithm 1 Write a List ls in Reverse Order

```
newList ← [ ]
i ← length(ls)−1
while i ≥ 0 do
    append ls[i] to newList
    i ← i − 1
return newList
```

This algorithm would have time complexity $O(n)$ since it would take one timestep to write down each element of the list and the rest of the operations do not scale with the size of the input[5]. The number n is in fact a polynomial of the number n, so this problem can be solved in polynomial time. It is in the class P. To give another example, if I had a list of length n and I wanted to make a table out of it with a row of n zeros for each entry of the list, the algorithm might look something like this:

Algorithm 2 Write a Table with a Row of Zeros for Each Entry in a List ls

```
table ← [ ]
for each item in ls do
    append an empty list to table
    for each item in ls do
        append 0 to the newly created list
return table
```

This algorithm has time complexity $O(n^2)$ since for each element in the n sized list, we write down n new table entries (and the sum of n groups of n is n^2), and the other operations don't scale with the size of the input. Similarly to the case above, we observe that n^2 is a polynomial of n. Therefore this task can be accomplished in polynomial time and the problem of creating an n by n is in the class P.

[5]When describing the runtime of an algorithm, it is conventional and generally useful to talk about its Asymptotic Time Complexity instead of its exact time complexity. Basically, this means we only worry about additional timesteps in the algorithm when they scale with the size of the input.

not exactly c
where I wa
put a definit
Big O notati
feels like it
be awkwar
put it right
in the mid
explaining w
is, but doing
fore talking
time comp
feels like I'm
the reader a
of inform
that they
no reason t
about yet.
I'll put it in
pendix and
reference it?
I don't know
this para
gets cut in
by the algo
but the te
definitely
the algorith
the .TeX file

There exists another class of problems called NP, which stands for Nondeterministic Polynomial time. These are problems that can be computed in polynomial time by a theoretical construct called a nondeterministic Turing machine,[6] or equivalently[7] (and for the purposes of this introduction, more expediently), questions for which we can determine whether a putative solution is correct in polynomial time. Since every problem in P can be *solved* in polynomial time by definition, we can infer that every problem in P is also in NP. But importantly, NP also contains problems which appear[8] to not be solvable in polynomial time. These apparently intractable problems are in a subset of NP called NP-Complete, which refers to problems that are in NP (i.e. we can determine whether a putative solution is correct in polynomial time), and are NP-hard[9], which means they are at least as hard as the hardest problems in NP. An example of a member of NP-Complete is the Subset Sum problem, which asks:

> For a set of numbers S, is there a non-empty subset of those numbers that sums to zero?

To show that this is a member of NP, I will give a polynomial time algorithm that verifies whether a putative answer is correct. For this problem, that means simply checking whether the candidate subset sums to zero.

Algorithm 3 Verify a Putative Answer for Subset Sum

```
total ← 0
for each number n in the given subset do
    total ← total + n
if total = 0 then
    return True
else
    return False
```

As you can see, this algorithm simply traverses the subset once, totaling up the values and returning True if they sum to zero and False otherwise. Since we look at each value in the subset once and the rest of the operations do not scale with the input size, this algorithm has time complexity $O(n)$. And of course, n is a polynomial of the input size n. So by the definition given above, Subset Sum is in NP. But the simplicity of the verifying algorithm can be deceptive. Don't be fooled. Subset Sum doesn't seem daunting at first, but a naive implementation of an algorithm to solve it requires checking every subset of S:

[6] PLACEHOLDER reference for reading up on nondeterminism

[7] PLACEHOLDER reference to proof showing that the nondeterminism definition and witness definition are equivalent

[8] Currently there is no proof which says whether or not P and NP are equivalent, but the hardest problems in NP have remained robustly intractable for quite a long time. Thus it *appears* that some problems in NP are not in P.

[9] If you're new to this terminology and this term feels poorly defined, don't worry. I explain the basis of NP-hardness and NP-Completeness in more detail later on. For now just follow the example.

Algorithm 4 Subset Sum of a Set S

```
for each subset of S do
    compute the sum of that subset
    if the sum is 0 then
        return True
return False
```

This algorithm computes the sum of every subset in S in order to see if any of them sum to zero. Since there are $2^{|S|}$ subsets of S, this algorithm has a time complexity of $O(2^n)$, which is *very* slow, and I'll give an estimate of just how slow it is using real world figures a little bit later on. But there are plenty of algorithms that have terrible runtimes if implemented poorly, and plenty of algorithms whose fastest known implementation is better than one might expect given the problem they are designed to solve. What makes us think that a polynomial time algorithm for Subset Sum simply hasn't been discovered yet? The answer is complicated. It is possible that there is a polynomial time algorithm for Subset Sum. There is no discovered proof that rules out this scenario. But the discovery of a polynomial time algorithm that solves Subset Sum would have profound implications and would cause an upheaval in Complexity Theory as we currently know it. I'll explain why.

Find a ref for proof: reduces to S Sum

There is a proof which demonstrates (using a method called reduction) that if we could solve Subset Sum in polynomial time, then we could solve another problem called 3-SAT in polynomial time. This problem, 3-SAT (short for 3 Conjunctive Normal Form Boolean Satisfiability), is readily convertible to a more general problem called SAT (short for Boolean Satisfiability), which is a very important problem in Complexity Theory. A foundational work called the Cook-Levin theorem proves that SAT is NP-Complete. In other words:

Reference Cook-Levin rem

1. SAT is in the class NP
2. **Any** problem in the class NP can be reduced in polynomial time to a SAT problem

The implications of this proof are profound. Fact **2** means that if we can do SAT efficiently, then in polynomial time we can convert any problem in NP to a SAT problem and solve it using our efficient SAT solving method. Thus if we could do SAT efficiently, we could do any problem in NP efficiently as well. More formally, if we had a polynomial time algorithm for SAT, we could convert any problem in NP to a SAT problem in polynomial time, and then solve that SAT problem in polynomial time. A polynomial + a polynomial = another polynomial, so the time complexity of this solution method would be polynomial. Thus, if we had a polynomial time algorithm for SAT, not only would SAT be a member of P, but (by the Cook-Levin theorem) ***every*** member of NP would be a member of P. Expressed compactly:

$$\exists \text{ a polynomial time algorithm which solves SAT } \implies \text{P} = \text{NP}$$

Now let's connect this observation with Subset Sum. As I stated above, there is a proof which demonstrates that if we could solve Subset Sum in polynomial time, then

we could solve a problem called 3-SAT in polynomial time. In other words, we can convert any 3-SAT problem to a Subset Sum problem in polynomial time. Further, a 3-SAT problem can be converted to a SAT problem in polynomial time. Thus if we wanted to solve any SAT problem, we could follow this algorithm:

Algorithm 5 Solve a SAT problem S via Subset Sum

Given a SAT problem S, convert the problem to a 3-SAT problem, name it T
convert T to a Subset Sum problem, name it U
solve U using a Subset Sum solving algorithm
return the result of solving U

Thus if we had a polynomial time algorithm for solving Subset Sum, all the pieces of this algorithm would take only polynomial time. And therefore we would have a polynomial time algorithm for solving SAT. Therefore by the Cook-Levin theorem P would equal NP. What I am describing is the basis of NP-hardness, the concept I glossed over earlier. NP-hardness is a generalized version of Fact **2** from the Cook-Levin theorem: a problem is NP-hard if any problem in the class NP can be reduced to an instance of that problem in polynomial time. In order to demonstrate that SAT is NP-hard, the Cook-Levin theorem had to do a lot of heavy lifting (e.g. reasoning about what it is *possible* for a nondeterministic Turing machine to do), but because we know that SAT is NP-hard, proving that other problems are NP-hard becomes much easier. We simply show that any instance of SAT can be reduced to an instance of the problem in question, and constructing an algorithm similar to the one shown above demonstrates that the problem in question is NP-hard. Any problem that is in NP and is NP-hard we call NP-Complete.

With NP-Completeness explained more extensively, we can begin to understand why the discovery of a polynomial time algorithm for Subset Sum would cause an upheaval in Complexity Theory as we know it, why we might expect Subset Sum (and all other members of the class NP-Complete) to be robustly inefficient. And any mathematician readers are going to have to forgive me for a moment because in the absence of a proof I'm going to have to use somewhat empirical ideas to explain this expectation. The question of whether NP-Complete problems can be solved in polynomial time (equivalently, the question of whether P is equal to NP) is widely considered the most important open question in computer science today. Formally introduced in 1971, the problem has remained open for quite some time now, despite a million dollar bounty on its head courtesy of the Clay Mathematics Institute. Any polynomial time algorithm for **any** NP-Complete problem (of which there are many many examples) would demonstrate that P = NP, but none have arisen in the 46 years since the description of the problem. Indeed, the fact that proving P=NP simply requires an example of an algorithm for an NP-Complete problem that runs in polynomial time makes it seem like the positive direction of the proof is somehow easier than the negative. It seems that since the method of a whole family of feasible proofs is obvious, if the positive direction were correct it seems more likely that we would have found it by now. The negative direction on the other hand is quite murky. How does one even begin to prove that P $\neq$ NP? Complexity theorists are for the

ook, Stephen
. "The
exity of
m prov-
procedures".
edings of
hird Annual
Symposium
Theory of
uting. pp.
3.
ed to cite
website that
aw at the
C&C but I
't find it by

most part just as baffled as you are for this particular question. For this reason and a few other empirical observations about the current state of Complexity Theory, our *expectation* is that P $\neq$ NP and therefore we expect that Subset Sum and Hypergraph Shortest Path do not have polynomial time algorithms. That expectation cannot be said to be *correct* until a proof arises, but in the absence of a proof we are stuck with algorithms that run in super-polynomial time. But just how bad is super-polynomial time complexity? I'll explain in concrete terms below.

All of the biologically applicable standard graph algorithms we discussed earlier (e.g. finding the shortest path from a start node to each other node, finding the strongly connected components in a directed graph, breadth first search) run in polynomial time. Computers can do them reasonably quickly, even on graphs of substantial size that we might encounter in the real world (e.g. a graph depicting components of a bunch of signaling pathways, a graph depicting who is friends with who on a social network, a graph depicting actors who have worked with other actors in films). The qualified attribute "reasonably quickly" means that if your graph is big enough and your algorithm's time complexity is slow enough (e.g. $O(n^3)$ or $O(n^2)$ instead of $O(n \cdot \log n)$ or $O(n)$), you might have to buy some computation from Amazon, but it will get done.

This is not true of any known algorithm that computes the solution to an NP-hard problem for all inputs (e.g. an algorithm that solves Subset Sum, an algorithm that solves Hypergraph Shortest Path). These algorithms run *unreasonably* slowly. That is to say, if the input graph is big enough, you can buy as much computation from Amazon as you want - the algorithm still might not be finished for years (and often much longer!). These algorithms have time complexity that is worse than polynomial with respect to the size of the input. A common and reasonable example of a super-polynomial time complexity is 2^n. This time complexity is common because 2^n is the number of timesteps it takes to make each possible set of binary choices for a set of n decisions, which is something you might often need to do to tackle a difficult problem. To convey an understanding of just how bad this is, I'll provide a hypothetical example using charitable, seat-of-the-pants calculations conducted using concrete figures from the real world:

> At time of writing, the highest clock speed (i.e. the rate at which a processor does an operation) offered from Amazon's cloud computing services is 3.3 gigahertz. In other words, the processor does something $3.3 \cdot 10^9$ times per second. That's pretty fast, and even though in the real world the processor is never devoting all of those ticks to one particular process, for this example let's be charitable and say that all of the ticks go directly towards running the algorithm. For $n = 100$, so for example that could be a graph with 100 nodes, a rough estimate of runtime an algorithm with time complexity n^2 would take $\frac{100^2}{3.3 \cdot 10^9} = 3.03$ microseconds, which Wolfram Alpha informs me is roughly twice the half-life of a muon (whatever that is, the point is it's very quick). For the same n, a rough estimate of the runtime of an algorithm with time complexity 2^n would take $\frac{2^{100}}{3.3 \cdot 10^9} = 1.217 \cdot 10^{13}$ average Gregorian years,

should pr
find some c
of a publish
per that dis
a possible
date proof m
for showing
NP
maybe cite
blogpost
C&C? Or
papers that
blogpost cite
these exampl
subject to c
will clean th
when I writ
'biological a
tions of grap
ory' section
intro

approximately **880 times the age of the universe**. And just to show that processing power isn't the problem, consider that even if we bought one hundred 3.3 GHz computers to run the problem concurrently, we're still looking at a computation time of roughly **9 times the age of the universe**. Clearly, this is a prohibitively expensive computation.

0.5 Parameterized Complexity: Practical Solutions to Theoretically Impossible Problems

0.6 Why use it?

LaTeX does a great job of formatting tables and paragraphs. Its line-breaking algorithm was the subject of a PhD. thesis. It does a fine job of automatically inserting ligatures, and to top it all off it is the only way to typeset good-looking mathematics.

0.7 Who should use it?

Anyone who needs to use math, tables, a lot of figures, complex cross-references, IPA or who just cares about the final appearance of their document should use LaTeX. At Reed, math majors are required to use it, most physics majors will want to use it, and many other science majors may want it also.

Chapter 1

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.

1.1 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.

1.1.1 References and Labels

Sometimes you'd like to refer to a table or figure, e.g. you can see in Figure 3.2 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 3.1 many factors can be assumed to follow from inheritance. Also see the Figure 3.1 for an illustration.

1.1.2 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

1.1.3 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.

1.2 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`)[2]. 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.

1.2.1 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.

[1] footnote text
[2] Reed College (2007)

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[3] 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[4] being different. Also the second time not in a row[5] should be different. Of course in other styles they'll all look the same.

1.3 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.

[3]Noble (2002)
[4]Noble (2002)
[5]Reed College (2007)

Chapter 2

Mathematics and Science

2.1 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.)

2.2 Chemistry 101: Symbols

Chemical formulas will look best if they are not italicized. Get around math mode's automatic italicizing by using the argument `$\mathrm{formula here}$`, with your formula inside the curly brackets.

So, $Fe_2^{2+}Cr_2O_4$ is written `$\mathrm{Fe_2^{2+}Cr_2O_4}$`
Exponent or Superscript: O^-
Subscript: CH_4

To stack numbers or letters as in Fe_2^{2+}, the subscript is defined first, and then the superscript is defined.
Angstrom: Å
Bullet: $CuCl \bullet 7H_2O$
Double Dagger: ‡
Delta: Δ
Reaction Arrows: $\longrightarrow$ or $\xrightarrow{solution}$
Resonance Arrows: $\leftrightarrow$
Reversible Reaction Arrows: $\rightleftharpoons$ or $\xrightleftharpoons{solution}$ (the latter requires the chemarr package)

2.2.1 Typesetting reactions

You may wish to put your reaction in a figure environment, which means that LaTeX will place the reaction where it fits and you can have a figure legend if desired:

$$C_6H_{12}O_6 + 6O_2 \longrightarrow 6CO_2 + 6H_2O$$

Figure 2.1: Combustion of glucose

2.2.2 Other examples of reactions

$NH_4Cl_{(s)} \rightleftharpoons NH_{3(g)} + HCl_{(g)}$
$MeCH_2Br + Mg \xrightarrow[below]{above} MeCH_2 \bullet Mg \bullet Br$

2.3 Physics

Many of the symbols you will need can be found on the math page (`http://web.reed.edu/cis/help/latex/math.html`) and the Comprehensive LaTeX Symbol Guide (enclosed in this template download). You may wish to create custom commands for commonly used symbols, phrases or equations, as described in Chapter 1.1.2.

2.4 Biology

You will probably find the resources at `http://www.lecb.ncifcrf.gov/~toms/latex.html` helpful, particularly the links to bsts for various journals. You may also be interested in TeXShade for nucleotide typesetting (`http://homepages.uni-tuebingen.de/beitz/txe.html`). Be sure to read the proceeding chapter on graphics and tables, and remember that the thesis template has versions of Ecology and Science bsts which support webpage citation formats.

Chapter 3

Tables and Graphics

3.1 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 3.1: 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.

Table 3.2: Chromium Hexacarbonyl Data Collected in 1998–1999

Chromium Hexacarbonyl			
State	Laser wavelength	Buffer gas	Ratio of $\frac{\text{Intensity at vapor pressure}}{\text{Intensity at 240 Torr}}$
$z^7P_4^\circ$	266 nm	Argon	1.5
$z^7P_2^\circ$	355 nm	Argon	0.57
$y^7P_3^\circ$	266 nm	Argon	1
$y^7P_3^\circ$	355 nm	Argon	0.14
$y^7P_2^\circ$	355 nm	Argon	0.14
$z^5P_3^\circ$	266 nm	Argon	1.2
$z^5P_3^\circ$	355 nm	Argon	0.04
$z^5P_3^\circ$	355 nm	Helium	0.02
$z^5P_2^\circ$	355 nm	Argon	0.07
$z^5P_1^\circ$	355 nm	Argon	0.05
$y^5P_3^\circ$	355 nm	Argon	0.05, 0.4
$y^5P_3^\circ$	355 nm	Helium	0.25
$z^5F_4^\circ$	266 nm	Argon	1.4
$z^5F_4^\circ$	355 nm	Argon	0.29
$z^5F_4^\circ$	355 nm	Helium	1.02
$z^5D_4^\circ$	355 nm	Argon	0.3
$z^5D_4^\circ$	355 nm	Helium	0.65
$y^5H_7^\circ$	266 nm	Argon	0.17
$y^5H_7^\circ$	355 nm	Argon	0.13
$y^5H_7^\circ$	355 nm	Helium	0.11
a^5D_3	266 nm	Argon	0.71
a^5D_2	266 nm	Argon	0.77
a^5D_2	355 nm	Argon	0.63
a^3D_3	355 nm	Argon	0.05
a^5S_2	266 nm	Argon	2
a^5S_2	355 nm	Argon	1.5
a^5G_6	355 nm	Argon	0.91
a^3G_4	355 nm	Argon	0.08
e^7D_5	355 nm	Helium	3.5
e^7D_3	355 nm	Helium	3
f^7D_5	355 nm	Helium	0.25
f^7D_5	355 nm	Argon	0.25
f^7D_4	355 nm	Argon	0.2
f^7D_4	355 nm	Helium	0.3
Propyl-ACT			

State	Laser wavelength	Buffer gas	Ratio of $\frac{\text{Intensity at vapor pressure}}{\text{Intensity at 240 Torr}}$
$z^7P_4^\circ$	355 nm	Argon	1.5
$z^7P_3^\circ$	355 nm	Argon	1.5
$z^7P_2^\circ$	355 nm	Argon	1.25
$z^7F_5^\circ$	355 nm	Argon	2.85
$y^7P_4^\circ$	355 nm	Argon	0.07
$y^7P_3^\circ$	355 nm	Argon	0.06
$z^5P_3^\circ$	355 nm	Argon	0.12
$z^5P_2^\circ$	355 nm	Argon	0.13
$z^5P_1^\circ$	355 nm	Argon	0.14
Methyl-ACT			
$z^7P_4^\circ$	355 nm	Argon	1.6, 2.5
$z^7P_4^\circ$	355 nm	Helium	3
$z^7P_4^\circ$	266 nm	Argon	1.33
$z^7P_3^\circ$	355 nm	Argon	1.5
$z^7P_2^\circ$	355 nm	Argon	1.25, 1.3
$z^7F_5^\circ$	355 nm	Argon	3
$y^7P_4^\circ$	355 nm	Argon	0.07, 0.08
$y^7P_4^\circ$	355 nm	Helium	0.2
$y^7P_3^\circ$	266 nm	Argon	1.22
$y^7P_3^\circ$	355 nm	Argon	0.08
$y^7P_2^\circ$	355 nm	Argon	0.1
$z^5P_3^\circ$	266 nm	Argon	0.67
$z^5P_3^\circ$	355 nm	Argon	0.08, 0.17
$z^5P_3^\circ$	355 nm	Helium	0.12
$z^5P_2^\circ$	355 nm	Argon	0.13
$z^5P_1^\circ$	355 nm	Argon	0.09
$y^5H_7^\circ$	355 nm	Argon	0.06, 0.05
a^5D_3	266 nm	Argon	2.5
a^5D_2	266 nm	Argon	1.9
a^5D_2	355 nm	Argon	1.17
a^5S_2	266 nm	Argon	2.3
a^5S_2	355 nm	Argon	1.11
a^5G_6	355 nm	Argon	1.6
e^7D_5	355 nm	Argon	1

3.2 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:

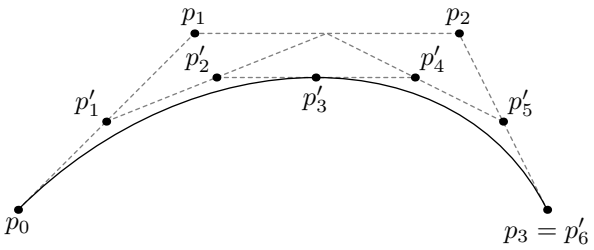

Figure 3.1: A Figure

3.3 More Figure Stuff

You can also scale and rotate figures.

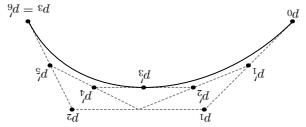

Figure 3.2: A Smaller Figure, Flipped Upside Down

3.4 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.

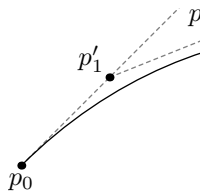

Figure 3.3: A Cropped Figure

3.4.1 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`.

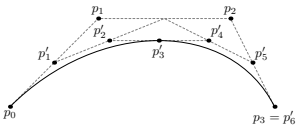

Figure 3.4: Subdivision of arc segments. You can see that $p_3 = p'_6$.

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.

Appendix A

The First Appendix

Appendix B

The Second Appendix, for Fun

References

Angel, E. (2000). *Interactive Computer Graphics : A Top-Down Approach with OpenGL*. Boston, MA: Addison Wesley Longman.

Angel, E. (2001a). *Batch-file Computer Graphics : A Bottom-Up Approach with QuickTime*. Boston, MA: Wesley Addison Longman.

Angel, E. (2001b). *test second book by angel*. Boston, MA: Wesley Addison Longman.

Deussen, O., & Strothotte, T. (2000). Computer-generated pen-and-ink illustration of trees. *"Proceedings of" SIGGRAPH 2000*, (pp. 13–18).

Fisher, R., Perkins, S., Walker, A., & Wolfart, E. (1997). *Hypermedia Image Processing Reference*. New York, NY: John Wiley & Sons.

Gooch, B., & Gooch, A. (2001a). *Non-Photorealistic Rendering*. Natick, Massachusetts: A K Peters.

Gooch, B., & Gooch, A. (2001b). *Test second book by gooches*. Natick, Massachusetts: A K Peters.

Hertzmann, A., & Zorin, D. (2000). Illustrating smooth surfaces. *Proceedings of SIGGRAPH 2000*, *5*(17), 517–526.

Jain, A. K. (1989). *Fundamentals of Digital Image Processing*. Englewood Cliffs, New Jersey: Prentice-Hall.

Molina, S. T., & Borkovec, T. D. (1994). The Penn State worry questionnaire: Psychometric properties and associated characteristics. In G. C. L. Davey, & F. Tallis (Eds.), *Worrying: Perspectives on theory, assessment and treatment*, (pp. 265–283). New York: Wiley.

Noble, S. G. (2002). *Turning images into simple line-art*. Undergraduate thesis, Reed College.

Reed College (2007). Latex your document. `http://web.reed.edu/cis/help/LaTeX/index.html`

Russ, J. C. (1995). *The Image Processing Handbook, Second Edition*. Boca Raton, Florida: CRC Press.

Salisbury, M. P., Wong, M. T., Hughes, J. F., & Salesin, D. H. (1997). Orientable textures for image-based pen-and-ink illustration. *"Proceedings of" SIGGRAPH 97*, (pp. 401–406).

Savitch, W. (2001). *JAVA: An Introduction to Computer Science & Programming*. Upper Saddle River, New Jersey: Prentice Hall.

Wong, E. (1999). *Artistic Rendering of Portrait Photographs*. Master's thesis, Cornell University.