Instructor Guide
for
Building Blocks, for version 1.3

Margaret M. Fleck

July 24, 2013

Chapter 1

Introduction

Like the textbook, this guide is intended to be succinct. I'll assume that you know how to flesh out the basic ideas in this guide using your own favorite style of presentation, using examples from a range of sources (including what we have on-line at Illinois), and so forth.

1.1 Our students

This book was designed for students with one term of programming and one term of calculus. We make very little direct use of calculus, but it helps to ensure an acceptable level of fluency with pre-calculus. There is a large change of topic and emphasis between this course and the calculus sequence. So some students have acceptable grades in the immediate prerequisite courses but still prove to have poor fluency with pre-calculus and/or simply find the proof material very difficult. See Section **??** below for suggestions about where these weaker students need help.

The past few decades have seen significant changes in the K-12 and early college math curriculum. Proofs have been de-emphasized, especially in the calculus sequence. But they are more likely to have already seen informal versions of Venn diagrams, summations, and counting formulas. They are experts at tracking down facts on the web, so we must be more creative when constructing homework problems, but textbooks no longer need to

compulsively include all relevant information.

This book is primarily aimed at students who are interested in applications, not foundations. In other words, students who are often dramatically different from what you were like in college. Part of the challenge teaching this material is to put yourself in the shoes of your former classmates, especially the ones who struggled to survive their theory courses. We need to devote significant time helping them understand basic ideas that you found obvious.

When teaching the small minority of CS majors with a strong interest in mathematical foundations, you will want to supplement this book with materials aimed at young theoreticians. See the bibliography for some starting suggestions. Also mathematics departments typically offer courses for math majors that would interest them.

Because we're going light on the foundational side, we introduce some concepts before formally defining them. In some cases, e.g. the real numbers, we never define them. (CS theory courses rarely do.) In the case of some other concepts, e.g. equivalence classes or the irrationality of $\sqrt{2}$, the concept is introduced early but the proof details are delayed until they have enough of a grip to appreciate them.

1.2 General approach

Because most of our students find proofs and logic challenging, this book spreads the difficult proof and logic concepts out over the entire term. The idea is to introduce these pieces at a measured pace, so that students are able to fully absorb each piece as it appears. Especially in the early sections it takes self-discipline to present only a simple account, without bringing up the more difficult issues.

In particular, many students require considerable practice to learn to write straightforward direct proofs. Until they have good fluency with direct proof, they aren't able to successfully build more complex proof outlines such as induction. Introducing proof by contradiction too early creates confusion about logical order. Moreover, many students write broken or complex contradiction proofs when direct proof or a counter-example would have been

simpler. So we delay presenting contradiction until chapter 17.

Other key logic/proofs topics moved later include vacuous truth (in Chapter 5), nested unlike quantifiers (Chapter 7), and proving an equality via two inequalities (Chapter 10). Circuits and logic have an obvious relationship but the main challenges in building elementary proofs involve concepts that are not shared with elementary circuit presentations: quantifiers and if/then statements. So the connection between logic and circuits is left until Chapter 16 (NP).

For related reasons, the model proofs in the book and lecture use very straightforward, somewhat verbose, versions of standard outlines. That's probably not your natural style. It isn't even mine. And the students won't write that way in later classes, when they get more fluent. But a simpler style is helpful to beginners.

We also cover a wide range of structures that are not especially difficult for the students, except insofar as we ask them to do proofs. These structures are introduced relatively early. This lets us use a more interesting range of examples to illustrate proof and logic ideas. Some of these structures occur in a very wide range of variations and applications. To avoid overwhelming students with "laundry list" lectures, we present only a sampling of the possibilities and spread them over a number of chapters. Coverage of graphs, in particular, is spread out over much of the book.

Chapter 2

Topics by Lecture

Material from the textbook divides roughly into three categories.

- The key ideas, the central topics for lectures.
- Routine definitions and examples: we have students read much of this on their own, with the aid of on-line drill problems.
- Harder examples and concepts, which need attention in lecture.

I'll discuss the content primarily in terms of lecture content. But it follows naturally that key lecture topics would then be practiced in discussion sections, with on-line drill problems, and on homeworks.

Because it's easiest to do this in concrete terms, the following is based on my lectures at U. Illinois. You should adapt this as necessary. Each of our lectures is 75 minutes long and there are two lectures per week. Our term has 29 lecture periods, of which two are eaten up by midterm exams and one is often lost to some misadventure (e.g. weather, loss of power).

The first third (about 11 lectures)

The first third of the course, Chapters 1-10, is about direct proof. Students learn to understand basic mathematical and logical constructs. They learn

to correct interpret definitions involving logical operators, notably quantifiers and implication. And they learn to get from assumptions to conclusions in logical order.

This basic material tends to seem obvious to instructors. And the top part of the class will pick up on the methods quickly. However, much of the class finds these basic methods challenging. So we take the material slowly, using a variety of topic changes to keep them practicing the same underlying skills.

Introduction (1 lecture)

The first lecture tends to be heavily concerned with administrative matters, which vary heavily with the term and the instructor. The first lecture might also give an overview and motivation.

Chapter 1 (Math review) contains material that should be largely familiar and, when not familiar, easy for properly prepared students. We have students read this on their own during the first two weeks and use on-line problems to help identify students with insufficient preparation.

Chapters 2-3: Logic and Proofs (3 lectures)

The goal is to cover only basic notation and concepts of logic plus basic direct proofs. So these two chapters should take only about 3 lectures. Don't get bogged down. Over the course of many chapters, we'll get more practice with these basics and also return to examine difficult topics.

These topics might break down into lectures as

- Lecture 1: Propositional logic
- Lecture 2: Predicates and basic proof outlines
- Lecture 3: More examples of proofs

Key topics for lecture 1 include the meaning of if/then and mechanical construction of negations and contrapositives. Also emphasize using precise mathematical English to make things readable: students tend to overuse shorthand. Much of the rest of propositional logic is well-handled by self-study drill problems. Mechanical drill problems are an excellent way to force students to pay attention to the (fairly easy) details of mechanical negation and contrapositive formation, which they need to have memorized to succeed later in the term.

The main focus of lecture 2 is a *basic* account of quantifiers and the four cases (Section 3.7) of how to prove/disprove quantified statements. Don't neglect (counter-)examples: even strong students often try to build general arguments or abstract examples where a simple very concrete example would suffice. In working through examples, include examples of forming negations and contrapositives of statements with quantifiers, recapping and extending the topic from lecture 1. Also emphasize how to apply definitions: using the definition in the form given, introducing fresh variable names when needed to avoid aliasing.

Notice that finding counter-examples is not the same skill as mechanical negation. When asked to negate a statement and also find a counter-example to it, students give answers that are only loosely coupled.

The nominal topics of Lecture 3 are two minor variations on direct proof: proof by cases, and proof by contrapositive. The more important hidden agenda is to re-iterate the basic methods introduced in lecture 2, using longer and more complex examples, and covering any small points that got missed in lecture 2. Because we are starting to see longer examples, we also introduce the idea of constructing proofs in two drafts.

Previous classes require students to show work, but do not emphasize readable presentation or logical order. In fact, they apparently teach a result-checking method which is basically backwards from logical order. Now and for the next several chapters, it is critical to emphasize setting up the starting assumptions for the proof and the desired end goal of the proof, each time you do a proof, to help them get a clear grip on the required order for their steps. Your lecture examples should model what you want their submissions to look like, e.g. full written-out connector words.

In previous math classes, students have typically written out their answers

in one draft, partly because they found the material easy and partly because clear presentation was not emphasized. So they must be told explicitly about using two drafts, using scratch paper, and/or writing the goal of the proof at the bottom of the page, leaving space to fill in the connecting steps later. Model these construction methods as you build proofs in lecture, now and for the rest of the term. Avoid writing the goal right after the assumptions are introduced (e.g. "We need to show XXX") because this muddies the logical order at a point when some students are very easily confused. Put the goal at the bottom of the page or over to the side or on your scratch paper.

Chapter 4: Number theory (2 lectures)

This chapter is primarily an excuse to practice our proof techniques from Chapter 3. Students need the practice. So the number theory content is kept fairly simple. We cover this in two lectures.

- Lecture 1: divides, remainder, and the Euclidean algorithm.
- Lecture 2: congruence mod k, modular arithmetic, and equivalence/congruence classes.

Some important methods to model as you work examples. The first three are review from earlier; the last is new.

- building proofs by working from both ends towards the middle
- applying definitions as written
- disproving conjectures with concrete counter-examples
- testing conjectures and exploring definitions by plugging small values into their defining equations

The Euclidean Algorithm is a convenient organizing focus for Lecture 1. The lemmas required to prove it correct are good examples of proofs using the definitions of divides and remainder.

Lecture 2 needs to cover congruence mod k, include an example proof using its formal definition. It introduces equivalence classes. Keep this pre-formal: we're just getting them used to the idea of treating a group of objects as a single object. Formal details will be done in Chapters 6 and 18.

In lecture 2, we also learn to do practical computations with modular numbers. It's fun to do some examples where the naive method would produce extremely large intermediate results, but smarter approaches can keep all the numbers small. E.g. computation of large powers by repeated doubling.

Chapter 5: Sets (1 lecture)

This chapter contains a lot of routine material, already somewhat familiar to students. This is a good application for on-line drill questions and does not require much attention in lecture. The big new ideas, which require about one lecture, are

- set-builder notation
- vacuous truth
- proving a subset inclusion

A bit of lecture time can also be devoted to common confusions with set notation and the recap of proof by contrapositive (section 5.13).

In this chapter, we keep the notational and conceptual issues simple by never nesting sets inside other sets. (Nested sets will be covered in Chapter 18.) So they tend to have problems such as confusing $\subseteq$ with $\in$, or 3 and $\{3\}$, or $\{7, 12\}$ with $(7, 12)$. And with set-builder notation, which they find somewhat complicated. It is probably their first exposure to the method of generate and test.

Vacuous truth is well known to be mysterious and counter-intuitive. They will feel confused and need to know that they are not alone in their confusion. Do not expect full understanding immediately. We will spend more time practicing with vacuous truth in Chapter 6.

Students are curiously reluctant to prove a subset relationship by the intended method: choose an element from the smaller set and show it lives in the larger set. Try to use concrete examples (e.g. as in section 5.10) so that the proofs look less like abstract nonsense. Tell them directly what outline they must use, otherwise they will make ill-advised attempts to improvise outlines ad hoc.

At this stage, it does not work to ask them to prove a set equality via two subset inclusions. This requires applying two distinct ideas, neither one of which is easy for them. Many students simply give up on the intended outline. So, right now, concentrate on getting them to do the proof correctly in one direction. Forcing equality via bounds from two directions will be covered in Chapter 10.

Chapter 6: Relations (1 lecture)

We discuss only relations on a single set. To keep the emphasis on the ideas rather than the notation, we make heavy use of pictures, i.e. treating these relations as directed graphs. We use on-line drill to get students on top of the easier relations concepts (e.g. reflexive). The single lecture revolves around three ideas that we have seen previously:

- vacuous truth
- proof by contrapositive
- two differently named variables might be equal (aliasing)

These ideas help us focus an analysis of the two definitions that cause the most conceptual difficulty and involve the most interesting proofs: transitivity and antisymmetry. We practice testing these properties and doing proofs on example relations.

One challenge here is that many students are still shaky on writing direct proofs, particularly the strategy of writing both ends first and then working to bridge the gap in the middle, Keep explicitly modelling the process of building the proof, not just your finished in-order product. A strong grip on

these skills now will make the later parts of the course much more pleasant for everyone.

Another challenge is that students are reluctant to try the recommended outline for a proof of antisymmetry. One issue is probably that they find it odd to set up two named variables that will turn out to be equal. It's important that they get comfortable with this pattern, because this aliasing idea is also used when manipulating equivalence classes, when proving a function to be one-to-one, and when manipulating pointer-based data structures.

It's good to do an example of a full equivalence class proof, i.e. including the extremely short subproofs of reflexive and symmetric. It's also good to include and talk through more examples of equivalence classes. We're still not doing all the formal details, e.g. partitions are left until Chapter 18.

Chapters 7-8: Functions (2 lectures)

Students think they understand functions from previous math classes. However, these classes have taught them a dangerously limited model of functions: real-number functions with succinct formulas. We need to upgrade this to a proper understanding of the wide range of possible functions.

First, we emphasize small discrete examples with no overt formula, e.g. using bubble diagrams, making it as hard as possible to try to work via formulas. We also make excuses to count the number of possible functions. We need them to understand that they can freely pair any output with any input and, therefore, that there are a vast range of possible functions. They need to appreciate the vastness of the possibilities in order to understand uncountability/uncomputability later.

Second, we must teach them to pay attention to type signatures and, especially, the declared input and output sets (domain and co-domain). Emphasize which set is which and emphasize the directions of arrows in bubble diagrams. Make clear the distinction between co-domain and image.

It takes about two lectures to cover functions, one lecture for each chapter. Main topics for the first lecture are

- What is "onto"?

- Understanding nested quantifiers

Nested quantifiers are well-known to be hard: don't count on full understanding from this first introduction. Because of this, teach the idea of "onto" by looking at whole sets (image vs. co-domain) rather than via the formal definition.

The second lecture covers

- what is one-to-one?
- pigeonhole principle
- compare set sizes by making functions

Proving a function one-to-one involves the same two ideas that we used for antisymmetric proofs in Chapter 6: contrapositive and aliasing. Using functions to compare the sizes of finite sets is an important first step towards understanding the later discussion of uncountability.

Pigeonhole proofs tend to be "trick" proofs, even if you try to select simpler examples. It helps to tell this to the students, so they don't get discouraged. On homework, it can also help to let them select (say) two problems from a set of three, in case they don't see the trick to one of them.

This early coverage of functions does not cover functions whose inputs or outputs are sets. Moreover, we don't cover the formal definition of a function as a set of pairs. We pick up these more difficult topics in Chapters 18 and 19.

Chapter 9: Graphs (1.33 lectures)

This chapter covers only the basics of (simple undirected) graphs. Undirected graphs are used as a continuing source of examples throughout the rest of the book. Directed graphs were introduced in Chapter 6 and will appear again in Chapter 19. But there is a wide range of graph concepts that we don't have time to cover: these make excellent source materials for class examples and homework problems.

The main hard concept is graph isomorphism, which takes about one lecture to cover. When we're comparing two graphs for isomorphism, they need to understand that the geometry of the graph picture doesn't matter, e.g. that vertices can be moved and edges bent. And, on the other hand, that edges need to move with the vertices they connect.

We typically ask them to count the number of isomorphisms between two graphs (easier), or from a graph to itself (harder). This requires understanding isomorphism and also organizing one's work to clearly describe the constraints involved.

We also look at how local features can be used to show that two graphs are not isomorphic or to constrain the search for isomorphisms. Common misunderstandings often involve the direction of the implications involved. For example, some students think that vertices of the same degree can be freely mapped to one another when, in fact, matching degrees is necessary but not sufficient.

This chapter also introduces a large number of unfamiliar terms. Notice that graph terminology is extremely unstable, so the main goal is not learning terminology but rather learning to properly interpret formal definitions. Online drill problems work well for the easier definitions towards the start of the chapter. However, it may be useful to spend some lecture time (e.g. a third of a lecture) working through examples involving paths and connectivity.

Chapter 10: 2-way Bounding (0.67 lectures)

Two-way bounding is an important technique that seems to be obvious to mathematicians but difficult for beginners to appreciate. In computer science, the most important application of upper and lower bounding is in algorithm analysis. However, algorithm examples are complex and lower bounds typically difficult to prove. So we draw most of our examples, particularly on homework and exams, from graph coloring, where both upper and lower bounds require only simple arguments. Therefore, this short topic combines well with the end of graphs.

There are two main ideas for lecture:

- proving an equality via upper and lower bounds
- sometimes upper and lower bounds don't meet

You need to convince students that both halves are important and that the two halves are different. Otherwise many will do only one half of these proofs. So select examples where the two halves of the proof require different techniques. Avoid examples (e.g. abstract set identities) where one half is basically a reversed version of the other and an adult would be tempted to replace the second half with "similar."

Even more experienced folks often to stop and think carefully to be sure of whether they have an upper or lower bound. Student have little experience using these terms in previous math and science classes. And, finally, they are uncertain about using these terms to describe bounds that aren't tight. So they also need practice (mostly outside lecture) with using the terms "upper bound" and "lower bound" on simple examples with familiar objects. For example, is 20 an upper bound on the age of a US president? a lower bound? neither?

The main goal for problems on graph coloring is to that they prove both the upper bound and the lower bound. In easy examples, the upper bound simply involves showing a coloring and the lower bound comes from a special subgraph (e.g. a wheel). Harder problems might require a case-by-case analysis to establish the lower bound. Or establishing how the chromatic number of a modified graph relates to that of the input graphs used to create it, e.g. what happens if we join two graphs with a cut edge.

To attack some of these harder examples, many students will need to be explicitly reminded that the choice of particular color names is arbitrary. So, for example, you can freely rotate the color space of one graph prior to joining it to a second graph.

The second third (about 7 lectures)

The second third of the course, Chapters 11-15, is about induction and recursion. Because induction and recursion are extremely important in computer

science but difficult for many students, we spread practice over several chapters:

- Basic inductive proof: Chapter 11
- With recursive definitions: Chapter 12
- Induction trees: Chapter 13
- Induction with inequalities: Chapter 14
- As applied to analysis of algorithms: Chapter 15

Chapter 11: Induction (2 lectures)

The main goal of this chapter is obvious: to write a straightforward proof by induction. We do this in two lectures.

The first lecture covers the basic idea and outline for an inductive proof. Although these simple examples require only weak induction, we consistently use strong inductive hypotheses, so that the form of the hypothesis remains stable. Emphasize the outline and name key components: P(x), inductive hypothesis, conclusion of the inductive step, closed form (where applicable).

The second lecture covers examples that require strong induction, i.e. reaching back more than one step. It also introduces an example of a "greedy" algorithm, for graph coloring. We also recap the idea from Chapter 10 that an upper bound might not be tight. Emphasize this: many students misremember this bound as an exact result.

Well-prepared students have done some recursive programming before this class. Also, we introduce induction after well-prepared students have mastered direct proof. So students typically get on top of the main ideas and outline with modest amounts of help. So much of our time is spent debugging the technical details.

Many problems center around writing a correct inductive hypothesis which connects properly to the rest of the inductive step.

- They may use the wrong choice of quantifier.
- They may use a single variable in a hypothesis that requires two, e.g. $k = 0, \ldots, k$.
- The variable used in the conclusion of the inductive step may match the wrong one of the variables used in the hypothesis.
- The conclusion of the inductive step may be for a slightly higher or lower value of the moving variable than the correct target.
- They may use $\leq$ where $<$ is required, or vice versa.
- They may start the hypothesis at the first value **after** the base case, rather than at the first base case.
- They may use a weak hypothesis in proofs which require a strong one. Or they may use a modified weak hypothesis involving the last two values of n.

When writing the base case

- They may write a chain of equalities that presupposes the claim they are trying to prove, e.g. that the lefthand side is equal to the righthand side which is equal to some formula.

Chapter 12: Recursive Definition (about 1 lecture)

This chapter continues practicing induction, gradually widening the space of examples. There are three main topics for lecture:

- How to read a recursive definition (should be easy)
- Using unrolling to find a closed form
- Writing an inductive proof for a result that is based on a recursive definition

Divide and conquer relations are central to computer science. However, at this level, floors and ceilings distract from the main point of the exercise. So we make simplifying assumptions to avoid them, e.g. find the closed form only for powers of 2 and assert without proof that the solution is monotonic between these selected inputs. Those continuing in the field will eventually take an algorithms course that will clean up these details.

Notice that unrolling, as a method, is prone to small algebra errors. This is true even when done by folks with solid algebra skills, e.g. course staff. Advise them to be careful, tell them you will be nice about small bugs in grading, and advise checking alleged closed forms for some small input values. Be mellow and let them have fun helping find out what stupid mistake **you** made while doing unrolling in lecture.

Chapter 13: Trees (about 2 lectures)

Basic definitions of trees and tree terminology are easy and can largely be taught with online drill. It's more helpful if lectures present some of your favorite motivating examples that illustrate why trees are useful in a wide range of applications, as well as covering the three harder topics:

- context-free grammars (lecture 1)
- tree induction proofs (both lectures)
- closed forms via recursion trees (lecture 2)

For this topic, the textbook adopts a distinctly applications-oriented perspective. A "tree" is, by default, a rooted tree with left-to-right order, matching applications usage. (See Chapter 21 for examples of "free", i.e. rootless trees.) Tree nodes typically contain labels, to create a wider range of interesting examples (e.g. for homeworks). And context-free grammars are presented as a way to define trees (as in AI or programming languages) not as a way to define sets of strings (as in theoretical treatments).

Context-free grammars are introduced partly because they are important in CS applications and partly to create a wider range of examples for tree

induction. Keep your presentation simple and concrete. Students will get lost if you use the more complex, more abstract treatments that are typical in later theory courses. Our goal is not to replace the automata theory course but to inspire students to take it.

Notice that our definition of context-free grammar is slightly non-standard. It's easy to show that the languages generated are the same as always. However, a wider range of tree shapes is allowed, to better cover the full range of practical examples.

The key to success in doing tree induction is to divide a tree at its root. If students adopt this approach, tree induction is straightforward. However, this requires convincing students not to use the method that many seem to naturally prefer: grafting extra nodes onto the leaf level. It works best to directly stipulate the divide-at-root method, because very few students can accurately assess whether supposed proofs using the grafting method are correct. (Sometimes they do work.)

Recursion trees are not very difficult conceptually. However, they require a lot of lecture time because the details are complex and easy to mess up. Try to keep enough focus on the main structure of how features of the tree (e.g. branching factor) relate to features of the recursively defined function (e.g. number of recursive calls). We cover only very simple examples of recursion trees, typically assuming input values that will make them full and complete.

Chapter 14: Big-O (about 1 lecture)

The main learning goals of this chapter are

- Good understanding of the dominant term method for working out big-O relationships, and
- Practice writing inductive proofs with inequalities.

The slightly edgy theoretical development is used so that we have a way to state (rather than merely imply informally) when a term is asymptotically smaller and therefore can be ignored. This allows us to derive a version of

the dominant term method that works properly even in the case of negative terms. Do the theoretical development with a light hand and don't expect students to be able to reproduce the supporting details. Keep the main focus on our end goal: the dominant term method.

In particular, do not attempt to provide a formal definition of a limit. In recent years, many calculus classes have de-emphasized formal definitions, apparently because so many students have memorized them without much understanding. An informal understanding of limits, which calculus classes do reliably provide, should be entirely sufficient for our purposes.

To help them understand the definition of big-O, we ask students to find suitable values for c and k. However, we don't ask them to write formal proofs of big-O relationships using this definition. In the vast majority of cases, it's simpler to rely on the dominant term method.

This chapter includes the final step in our multi-week approach to practicing inductive proof construction. Our students know the rules for manipulating inequalities. However, since previous math courses have done very little with them, most students lack fluency. This makes all proofs with inequalities, but especially inductive ones, more difficult than the corresponding ones with equalities.

Chapter 15: Algorithms (about 1 lecture)

This chapter presents three patterns of algorithm analysis:

- nested loops
- while loops (resource consumption)
- recursive

Students find nested loops straightforward. Resource consumption is a bit of a black art. So our main focus is on analysis of recursive algorithms. Specifically, our main job is to write the recursive running time definition corresponding to a code fragment. Techniques from previous chapters can then be used to find a big-O version of its closed form.

We emphasize very standard design patterns, whose analyses are worth memorizing. Karatsuba's algorithm is presented at the end to show that the range of possibilities is much larger. However, we frequently skim past many of its details and certainly don't expect students to reproduce them.

Many students have already seen standard array-based sorting and searching algorithms in their first programming course, and they will see them again in a later algorithms course. To prevent boredom, it's good to look for other types of examples, e.g. involving graphs or 2D geometry. Since we are teaching algorithm analysis, not algorithm design, it's ok to use sub-optimal algorithms as examples.

However, notice that students have seen very few data structures and some of these (e.g. linked lists) perhaps very briefly. So it's critical to keep pseudo-code simple and be very clear about implementation assumptions.

The final third (8 lectures)

The earlier parts of the course taught them techniques that they should be able to apply with some confidence as they move on to the next course. The final third of the course, Chapters 16-21, covers topics that are more abstract. We are primarily giving them a sense of the fun topics that they will learn in future theory courses and preparing them to understand these topics more easily when they do see them again. Therefore, in many cases, we expect students to come away with only a shallow understanding of subjects that are quite deep.

Some of these topics would traditionally be presented earlier in the course, e.g. the material in Collections of Sets. However, we've found that many students are not prepared to absorb these harder concepts until late in the term.

Chapter 16: NP (about 1 lecture)

This presentation of NP is intended to be very brief and introduce students to a central piece of CS theory culture. Essentially all the technical details

are beyond the scope of this course, so the goal is to make the main ideas accessible. To do this, we define as few abstract technical terms as possible. We also choose versions of problems (e.g. marker making) that relate well to applications rather than the simplified versions (e.g. bin packing) common in theory discussions. Students won't be writing proofs in this class and it's more important that they should understand why the ideas are important.

This is also a good place to discuss how logic relates to circuits. Logic circuits fit naturally into a discussion of NP. And the students now have enough mathematical maturity to absorbe abstract discussions of the relationship.

Chapter 17: Contradiction (about 1 lecture)

The basic outline of proof by contradiction is easy to understand. However, earlier in the course, students lack the maturity to use this method well. Rather than pushing to debug their direct or contrapostive proof, they write a contradiction proof with a complex, meandering outline that is frequently buggy. Or, even sadder, many try to use a broken multi-paragraph contradiction proof in place of a 1-line concrete counter-example.

We delay teaching contradiction so that they will be forced to get on top of the simpler methods that normally lead to easier, clearer proofs. Contradiction is then presented as a method to use selectively in two situations:

- When direct methods fail, as in the diagonalization proof we'll see in Chapter 20, or
- When the contradiction proof is much simpler, e.g. the classic proof of the irrationality of $\sqrt{2}$.

Chapter 18: Collections of Sets (about 1.25 lectures)

Many of our students find examples with nested sets very tricky. So we delay this topic until they have had considerable practice with more basic versions

of sets, functions, and so forth. And we focus almost entirely on getting them to understand sets with one layer of nesting: a set containing sets, each of which contains atomic objects. Except for a few throw-away examples, we don't use deeper nesting and we don't build sets containing a mixture of sets and atomic objects. Finally, we make set-containing sets more distinct by calling them "Collections" and using script variable names.

The first lecture reviews basic notation and then covers two difficult concepts:

- Set-valued functions
- Partitions

The difficulty here lies in the abstract nature of the concepts and notation. So take it slowly in lecture, emphasizing how each piece of notation translates into its informal meaning. Also emphasize the type of each example object, e.g. is X a student? a set of students? or a set of sets of students?

The second (about 25%) lecture covers the counting formulas from this chapter. Much of this material is semi-familiar to the students and can be taught using on-line exercises. The exception is the formula for combinations with repetition. Make sure they clearly understand the objects and separators picture used to derive this formula.

Chapter 19: State Diagrams (about 1.75 lectures)

State diagrams are a modest generalization of finite automata, intended to cover the full range of similar-looking examples found in computer science applications. They are also a nice example of directed graphs, which we saw briefly in the context of relations. As with Chapter 13 (Trees), this chapter takes an aggressively applications-oriented perspective. We're not trying to cover the kinds of formal details about finite automata typical of later courses. Rather, we're trying to show the big ideas with a range of practical applications, largely drawn from Artificial Intelligence.

Lectures can be divided as

- Lecture 1 (75%): basic notation and examples
- Lecture 2: representing transition functions and managing huge state machines

Students can very quickly pick up the basic notation and design ideas for these state machines. E.g. design a phone lattice that recognizes a simple class of strings, with simple uses of loops.

Transition functions are defined to return sets of states. This is partly for flexibility, e.g. when constructing phone lattices. But it also gives an excuse to keep practicing the baffling notation for set-valued functions. Students at this level lack the programming maturity to see why this non-determinism might cause headaches. So if you act like it's not a big deal, they won't worry.

The input/output pair representation of functions is traditionally presented as the formal definition of functions. This is unsatisfying to the students because it's dry and technical. It's unsatisfying intellectually because it's not really a definition: functions and sets of pairs aren't the same thing except when doing foundations. This book re-casts the discussion as a problem in storing functions inside a computer program, which avoids the philosophical issues and is of more practical relevance to the students.

Finally, we give a brief introduction to some issues presented by huge state machines: shared states (aka dynamic programming), on-demand state creation, and infinite state spaces. We don't expect students to do much, if anything, with these ideas. They are just to expand their cultural knowledge. And the Game of Life examples are reliably fun to present in class.

Chapter 20: Countability (about 2 lectures)

This topic is well known to be difficult. At this stage in their education, it's presented largely for general knowledge and to "prime the pump" for seeing this again in a future theory class. Notice also that we're getting close to the

end of the term. Students are getting burnt out. And you may be past the cutoff for the last topics that can be practiced on a homework.

So, don't expect much in terms of students applying these techniques themselves right now. Relax and try to keep it fun.

Cantor Schroeder Berstein's theorem is used more extensively than is traditional. This is partly because it allows very simple, non-messy proofs of countability. But, also, it gives us another opportunity to demonstrate the useful 2-way bounding technique from Chapter 10.

This topic can be covered in one lecture. However, it's more comfortable to use two lectures, so that tired students have a chance to think about the ideas and ask questions.

In a 2-lecture presentation, the first lecture might cover

- measuring size of infinite sets via bijections and one-to-one functions
- proving equality two ways: direct construction of a bijection and via 2-way bounding
- simple examples of countable sets

The second lecture then covers uncountability:

- diagonalization proof
- examples of uncountable sets
- intuition: uncountable sets contain individual objects (sequences, subsets, tilings, program traces) that are aperiodic, i.e. have infinite length but no repeating pattern
- there are more functions than (a) succinct formulas or (b) programs (uncomputability)
- aperiodic tilings

Many students will find the diagonalization proof mysterious and unsettling. Reassure them that this is normal. It's a grossly non-constructive

proof of a counter-intuitive result. Remember that their models of the reals and the rationals are not crisp enough to provide much distinction between the two. Don't expect them to answer deep questions on this topic: save that for the next theory course.

Aperiodic tilings are a fun example to end on. They have very pretty pictures and a wonderful historical story. Who would think that abstract results about uncomputability would be so closely tied to Nobel-prize winning practical results in Chemistry? Also, they provide a concrete picture for the kind of behavior that makes the Halting Problem undecidable.

Chapter 21: Planar Graphs (optional extra)

This is a fun topic if you have extra time at the end of the term, or if you need supplementary material for honors students. It makes a good capstone topic, because it uses a range of techniques from earlier in the course, especially the inductive proof of Euler's formula. In its complete form, it requires two lectures, but you can hit the high points in a single lecture.

Conclusion (1 lecture)

The last lecture is typically consumed by wrap-up activities, e.g. filling out course evaluations, summarizing the course, previews of following courses, review for the final exam.

Chapter 3

Helping Weaker Students

The previous chapter discussed how to present the material for typical students who enter the course with the intended background. Inevitably, some students attempt to take this course with inadequate preparation. As a shorthand, I will refer to these as "weaker students" with the understanding that I'm describing only their current state rather than their long-term potential. This section contains suggestions about how to identify these students and, when possible, address their weaknesses so that they can succeed.

We will focus on describing the skills involved. Your situation and philosophy will determine how you choose among strategies such as:

- Persuading students to drop, e.g. and take another course first.
- Channelling students into a prep course.
- Providing extra drill problems.
- Providing extra discussions or tutoring.

If a student truly lacks the background to survive the course at this point, it is kindest to help them find a more appropriate course as early as possible rather than having them fail the course or drop halfway through the term. Similarly, lowering standards at this point sets the student up for a more damaging failure in later parts of their major. So problems with

preparation need to be made manifest and fixed. Because the main effect is overall mathematical maturity, poorly prepared students may be helped by technical courses outside the math sequence, such as programming and physics.

The problems detailed below are common in weaker students. However, they are also places where stronger students are a bit shaky, have to work carefully, and make occasional errors. So a modest amount of drill and review on these topics is appropriate for the whole class.

Chapter 1: Math Review

Students with poor preparation typically lack fluency with algebra and pre-calculus. So they exhibit the following sorts of problems. Often, they do know the right answer if given sufficient thinking time, but become inaccurate under time pressure or when these are merely a small step in solving a larger problem (e.g. writing a proof).

- Have difficulty doing simple problems with logs and exponents, even when reminded of easily-forgotten formulas (e.g. change of base).
- Have trouble doing algebra with symbolic fractions.
- Can't quickly factor 2nd order polynomials where trial and error works, e.g. all coefficients are integers. Or it doesn't occur to them to factor a polynomial when this might be useful, e.g. in constructing proofs.
- Treat integers, rational, reals as mutually exclusive categories, e.g. 3 isn't a real number.
- Test membership in standard sets (integers, reals, etc) using superficial features rather than simplifying and checking the right property, e.g. $\sqrt{2} \cdot \sqrt{2}$ is irrational.
- Believe infinity is an actual number.
- Make an undue number of mechanical mistakes in algebra derivations, e.g. $k+k+k+k = k^4$ or, more subtly, $5(3^{k-1} - 2^{k-1}) = 15^{k-1} - 10^{k-1}$.

Many students have problems with inequalities, both the underlying concept of a bound that isn't tight and the mechanics of manipulating algebra with inequalities. So they are all prone to the following sorts of problems, but they are more common with the weaker students.

- Confusion of $\leq$ and ¡, positive and non-negative.
- Negation of $\leq$ to $\geq$.
- Trouble doing algebra derivations involving inequalities.

Insufficient mathematical maturity also makes it difficult to quickly pick up easy concepts such as

- How to use summation notation, e.g. change the indexing, extract the first or last term, move a constant multiplier outside the summation.
- String notation

To succeed, students also need to have taken an introductory programming class. Ideally, this should have included manipulation of values in arrays, building simple recursive functions, and some exposure to linked lists or other pointer-based structures. Students without this background need to be warned explicitly at the start of term, because these skills won't be needed until later. Simple pseudo-code reading exercises can be useful for detecting and scaring folks who lack both this background and the ability to quickly pick up enough to get by.

Chapters 2-3: Proofs and Logic

Weaker students have trouble getting on top of the simple mechanical aspects of propositional and predicate logic. Where stronger students only need to work on the less obvious issues, such as how OR differs from XOR or how to negate AND, the weaker ones sometimes have trouble understanding how AND and IMPLIES differ in meaning, or even how AND differs from OR.

Weaker students also tend to believe that a statement is not true if it expresses less information than we know. For example, they think $|x-1| \leq 8$ does not imply $x \leq 100$ because we should have concluded that $x \leq 9$.

Weaker students may have trouble applying the definition of rational number. They may fail to simplify e.g. $\frac{\sqrt{2}}{\sqrt{2}}$ is irrational. They may not ensure that the numerator and denominator are integers, e.g. $\frac{1}{\sqrt{2}}$ is rational. They may assume that a fraction must be in lowest terms or not heavy, e.g. 3 isn't rational.

Chapter 4: Number Theory

Now that we're starting to write proofs of some complexity, using concepts that aren't entirely familiar, weaker students tend to get stuck trying to solve homework problems. They need additional help learning to use methods which have been presented in lecture:

- Building a proof by working inwards from both ends.
- Applying the definition of a key concept (e.g. divides) to narrow that gap.
- Testing a conjecture or exploring a definition by trying small examples

In the discussion of modular arithmetic and equivalence classes, they may need

- Additional practice doing modular arithmetic.
- Help understanding distinction between a fraction and the natural number it names.

Chapters 5-6: Sets and Relations

Aside from the hard new concepts discussed above, weaker students need drill problems to help them get on top of routine concepts and new notation.

For example, they may confuse the notation for a set and an ordered tuple, they often have extra or missing levels of set brackets, and so forth.

Their grasp of quantifiers and if/then statements tends to be shaky. They think they need more examples of what does/doesn't satisfy the relation properties, and those are indeed useful. But addressing their underlying issues should also involve exercises that work directly on their understanding of quantifiers and if/then, using more basic examples that don't involve relations.

Finally, they still need help with basic proof-building skills. They are even more reluctant than other students to prove antisymmetry by assuming xRy and yRx and showing that $x = y$.

Chapters 7-8: Functions

Weaker students often confuse a single value (e.g. the integer x), a set of values (e.g. $\mathbb{Z}$), and a type (e.g. integers). So they write confused statements like "17 is a $\mathbb{Z}$" or "17 is a domain." They also tend to confuse the domain and co-domain, especially when asked to write formal definitions using quantifiers. It is not always clear whether they have a broken picture in their head or a broken model of what the words mean and how to use them fluently. Mechanical drill probably helps here.

Where other students are struggling primarily with nested quantifiers, weaker students still have a hazy understanding of single quantifiers and the meaning of if/then. So they may confuse "some" with "all" in writing. Or they may avoid using quantifiers to express ideas such as one-to-one, prefering to use words like "unique" in ways which are often ambiguous. And they have trouble knowing precisely what would constitute a counter-example to a claim about one-to-one or onto.

Some of the confusion between domain and co-domain seems to be at the level of underlying concepts. When asked to draw the bubble diagram of a function that is not one-to-one, many weaker students draw an example where one input value corresponds to two output values. Diagram-based exercises are useful in testing and drill for students with weak ability to translate ideas into formal mathematical statements.

Chapters 9: Graphs

The weaker students have more difficulty getting on top of the details of the concepts. They have a spongier understanding of the mathematical jargon used to write the definitions. And they may be less good at paying close attention to detail as they read. So they need more drill on the basic definitions.

These students find isomorphism extremely difficult. Even once they understand the concept of isomorphism, they may show signs of poor general problem-solving skills. That is, trouble

- analyzing the constraints methodically,
- identifying features which tightly constrain the form of the solution, e.g. a vertex that is the only one with a specific degree,
- checking that a plausible vertex match (e.g. same degree) actually works when you try to line up the edges,
- giving up on a line of analysis that isn't working and trying something else,
- applying counting rules, e.g. 4 vertices that can be freely permuted generates 4 (in place of 4!) possibilities, and
- explaining their analysis clearly.

They have a tendency to apply simplified methods that don't work right.

Chapter 10: 2-way Bounding

Few students are fluent with bounds or even with numerical inequalities. Weaker students may have major problems with manipulating inequalities algebraically and learning to use the terms "upper bound" and "lower bound." They are likely to misremember whether a method (e.g. a wheel subgraph) yields an upper bound, a lower bound, or an exact result. They also forget

extremely quickly that they have to establish both bounds e.g. for graph coloring problems.

Frequently, weaker students did not get fully on top of set builder notation when we saw it earlier and/or absorbed it only superficially so they it has now been forgotten. So they may need review and further practice at this point.

Chapter 11: Induction (2 lectures)

Weaker students may not get the basic ideas behind induction and write proofs whose central supporting structure is wrong. For example

- They may state the values of n required for the base case, and perhaps compute one side of the equation involved in the claim, but not actually check the truth of the claim.
- Their inductive hypothesis may state that the claim is true for all values of n.
- Their inductive step may fail to use the inductive hypothesis at all, or perhaps use it only once in proofs where it must be applied twice.
- Their inductive step may prove the claim for a value one smaller, rather than one larger, than the largest value covered by the hypothesis.

These students may also have a weak grasp of logical order and writing a simple direct proof. So their inductive step may be more-or-less backwards.

Chapter 12: Recursive Definition

Weaker students may have difficulty with this topic because they have difficulty

- interpreting recursive definitions,

- manipulating summation notation, and/or
- switching between sequence notation and the nearly equivalent function notation.

They may also not yet understand the basic ideas and mechanics of induction.

They may also

- Have trouble figuring out how many base cases are required.
- Misinterpret the recursive definition as having proved the result for the base cases, and thus start their base cases at the next higher value.

Finally, limited fluency with logs, exponents, and summations may create challenges when they try to simplify unrolling results.

Chapter 13: Trees

Weaker students often have lingering problems reading simple formal definitions. So they may need more help (e.g. online drill) with the large number of tree definitions and basic facts. Check for

- special cases, e.g. trees of height zero
- correct use of similar terms, e.g. level vs height
- understanding how context-free rules map onto local tree constraints (e.g. left-to-right order matters)

Weaker students are likely to have problems relating features of a recursive definition to features of the corresponding recursion tree. This may be due, in part, to a weak grasp of recursive definition. But they can be helped a lot by a systematic, step-by-step approach to recursion tree construction.

Chapter 14: Big-O

Even for the weaker students, the main ideas of big-O aren't that hard. However, they may have substantial problems doing induction proofs with inequalities. It might help to offer optional review problems prior to this chapter.

Chapter 15: Algorithms

Weaker students may have trouble forming a clear picture of how the code fragments work. E.g. they may interpret recursive procedures incorrectly. Or they may have trouble figuring out how many times a recursive procedure is called. This is most common with students who took this course without the listed programming prerequisite or who did poorly in that course.

Weaker students also have problems mapping key features of the code to key features of the recursive running time definition. This is probably the same students who had trouble mapping recursive definitions to recursion trees. Again, systematic step-by-step tutoring (in person or online) can help.

Chapter 16: NP

We don't expect students to get a deep understanding of this topic, so weaker preparation causes few problems here.

Chapter 17: Contradiction

The first step in building a contradiction proof is to form the negation of the claim. Weaker students may still have trouble doing this correctly.

The end goal of contradiction proof is poorly defined. Weaker students may need help exploring several hypotheses for what might make a good final contradiction.

Finally, weaker students may still be relying on intuitive heuristics for logical order, rather than a crisp model of what steps are ok. Intuition works less well in proofs by contradiction, because most of the proof describes a counter-factual situation. They may need more help getting their working notes into proper logical order for their final draft.

Chapter 18: Collections of Sets

Weaker students may not be fully on top of basic set and function concepts, making them less able to follow the details of this extension. They may also have lost track of definitions presented earlier in the term, e.g. the equivalence class notation we used for modular arithemetic. Review exercises might be helpful here.

Weaker students also have difficulty mapping notation onto concepts. For example, if the co-domain of a numerical function f is $\mathbb{P}(\mathbb{Z})$, then what sort of object is $f(3)$? These students also need extra help with correct use of terminology. E.g. some may refer to any collection of sets as "a power set."

Finally, weaker students may be less good at close reading of notation. Reading problems may be helping make this class hard for them. But, also, poor understanding of the math may be making it harder to read. In either case, they might benefit from exercises that make them look closely at features of expressions such as the numbers of curly brackets.

Chapter 19: State Diagrams

The weaker students may not yet have a solid grasp of the ideas from the previous chapter, notably

- Manipulating collections of sets,
- Reading type signatures, especially those involving power sets.

They often get confused between the whole domain/co-domain and a single input/output value.

Weaker students may need help to understand the basic notation and design of simple automata. They may be less good at debugging their designs, either due to weaker programming skills or weaker proofreading skills or general overload at the end of the term.

Chapter 20: Countability

The weaker students will have a lot of trouble with this material, but it's not a big tragedy if they get only a very superficial picture. Like the rest of the class, it will be in the next theory class where it's critical that they follow all the details.

Weaker students may get more out of the proofs if they are helped with one specific technical issue: how to represent a set as a bit vector.

Chapter 21: Planar Graphs

This topic is optional and fairly lightweight. It's not a big deal whether the weaker students get fully on top of the details.