CS 103 (Mathematical Foundations of Computing) Course Notes

R. Sikand

Spring 2021

These are notes for Stanford's CS 103: Mathematical Foundations of Computing, instructed by Cynthia Bailey Lee with lectures by Keith Schwarz. The official website for a previous offering of the course is linked at http://web.stanford.edu/class/archive/cs/cs103/cs103.1202/. The first half of the course covers the mathematical foundations, such as discrete structures, whereas the second half offers an introduction to the theory of computation.

Contents

§1 Induction

Theorem 1.1 (Principle of Mathematical Induction)

Let P be some predicate. The principle of mathematical induction states that if

$$P(0) \text{ is true}$$

and

$$\forall k \in \mathbb{N}.(P(k) \to P(k+1))$$

then

$$\forall n \in \mathbb{N}.P(n)$$

Induction is a powerful technique that we can use–more specifically, in proofs. It is essentially saying that, for some predicate P, if $P(0)$ evaluates to true, and that for any k, if $P(k)$ also evaluates to true, then we know that $P(k+1)$ evaluates to true. More so, induction is a two step process: showing that $P(0)$ is true, then showing that $\forall k \in \mathbb{N}.(P(k) \to P(k+1))$ is true [1]. Notice that the second part involves an implication. This is important. Recall that to prove an implication, we assume that the antecedent is true (if...), and prove that if such is true, the consequent (then...) is also true. Now **here is the catch** (and the nifty part about induction): for the inductive step, we have presumably verified that the base case ($P(a)$) is true. Thus, from the inductive step, we see that $P(a+1)$ is true, which, also by the inductive step, we see that $P(a+2)$ is true and so on. Hence, the theorem is proved since that pattern continues forever and holds for all $a \in \mathbb{N}$ [4]. So even though it might seem like at first that we aren't picking some arbitrary number (which is what we'd usually need to do with universally-quantified statements), we pick a specific value (namely the base case), but then show that through our own inductive hypothesis, that a step greater than the base case is also true. This in turn means 2 steps greater is true which shows that 3 steps greater is true and so on. Again, this pattern repeats infinitately so we've proved the result for all values which would allow them to "pick any arbitrary $n \in \mathbb{N}$".

I think a good overall distillation of induction described in [2] is as follows: "If it starts true... and it stays true... then it's always true.".

It is also important to recall the truth table for implications:

p	q	p → q
T	T	T
T	F	F
F	T	T
F	F	T

Formally, we refer to the first part ($P(0)$) as the **Base Case** and refer to (proving) the second part ($\forall k \in \mathbb{N}.(P(k) \to P(k+1))$) as the **Inductive Step**. More so, in the implication, the assumption that $P(k)$ (the antecedent) is true, is called the **Inductive Hypothesis**.

[1] You might be tempted to simply take the base case (in this case, $P(0)$) and use that as your part of your proof when proving the inductive hypothesis. Remember however, that the definition of the inductive hypothesis uses a universal quantifier. Thus, you'd need to state to the reader that they can pick "any arbitrary value as their k".

So when should one use mathematical induction? To put it simply, [4] states that induction is used for proving statements that read along the lines of: "Suppose you want to prove a theorem in the form 'For all integers n greater than equal to a (base case), $P(n)$ is true'".

§1.1 Proof by Induction

Expanding off what was mentioned above, we can use induction to prove things in an elagent manner [2]. In proof by induction, we need to prove two things:

- That the base case is true.
- That the inductive hypothesis is true (proving the inductive hypothesis part is referred to as the **inductive step**).

To prove the base case, simply plug your value into the chosen predicate. The inductive step is a little trickier, but the main thing to takeaway is that you now just have an ordinary implication that you need to prove (using techniques like direct proof, proof by contradiction, proof by contrapositive).

Example 1.2

Theorem: The sum of the first n powers of two is $2^n - 1$.
First off, notice the pattern here:

$$\begin{aligned} 2^0 = 1 &= 2^1 - 1 \\ 2^0 + 2^1 = 1 + 2 = 3 &= 2^2 - 1 \\ 2^0 + 2^1 + 2^2 = 1 + 2 + 4 = 7 &= 2^3 - 1 \\ 2^0 + 2^1 + 2^2 + 2^3 = 1 + 2 + 4 + 8 = 15 &= 2^4 - 1 \\ 2^0 + 2^1 + 2^2 + 2^3 + 2^4 = 1 + 2 + 4 + 8 + 16 = 31 &= 2^5 - 1 \end{aligned}$$

Proof. Let $P(n)$ be the statement "the sum of the first n powers of two is $2^n - 1$." We will prove, by induction, that $P(n)$ is true for all $n \in \mathbb{N}$, from which the theorem follows. For our base case, we need to show $P(0)$ is true, meaning that the sum of the first zero powers of two is $2^0 - 1$. Since the sum of the first zero powers of two is zero and $2^0 - 1$ is zero as well, we see that $P(0)$ is true. For the inductive step, assume that for some arbitrary $k \in \mathbb{N}$ that $P(k)$ holds, meaning that

$$2^0 + 2^1 + \ldots + 2^{k-1} = 2^k - 1$$

We need to show that $P(k+1)$ holds, meaning that the sum of the first $k+1$ powers of two is $2^{k+1} - 1$. To see this, notice that

$$\begin{aligned} 2^0 + 2^1 + \ldots + 2^{k-1} + 2^k &= \left(2^0 + 2^1 + \ldots + 2^{k-1}\right) + 2^k \\ &= 2^k - 1 + 2^k \\ &= 2\left(2^k\right) - 1 \\ &= 2^{k+1} - 1 \end{aligned}$$

Therefore, $P(k+1)$ is true, completing the induction. □

[2] I'd like to say that this notion is in parallel with when one can use recursion instead of iteration to reach an intended result much faster in programming.

From this, we adapt to a generalizable proof by induction template:

Example 1.3

Template for Proof by Induction:

Proof. Let $P(n)$ be the statement <*insert predicate here*>. We will prove, by induction, that $P(n)$ is true for all $n \in \mathbb{N}$, from which the theorem follows. For our base case, we need to show $P(0)$ is true, meaning that <*insert intended result when plugging in base case into predicate*>. Since <*insert ground truth value of* $P(0)$> and <*insert* $P(0)$ *result to show equivalence with ground truth value*> as well, we see that $P(0)$ is true. For the inductive step, assume that for some arbitrary $k \in \mathbb{N}$ that $P(k)$ holds, meaning that <*show expanded result when evaluating* $P(k)$>.
We need to show that $P(k+1)$ holds, meaning that <*insert intended result when when evaluating* $P(k+1)$>. To see this, notice that <*insert inductive step intuition (possibly some algebraic manipulation); the goal is to show that the actual result when evaluating* $P(k+1)$ *is equivalent to the intended result as explained in the previously sentence*>.
Therefore, $P(k+1)$ is true, completing the induction. □

§1.1.1 Structuring a Proof by Induction

In words, a proof by induction is structured as follows (note that the following was taken from [2]):

- Define some predicate P that you'll show, by induction, is true for all natural numbers.
- Prove the base case: State that you're going to prove that $P(0)$ is true, then go prove it.
- Prove the inductive step:
  - Say that you're assuming $P(k)$ for some arbitrary natural number k, then write out exactly what that means.
  - Say that you're going to prove $P(k+1)$, then write out exactly what that means.
  - Prove that $P(k+1)$ using any proof technique you'd like!

The power of proving by induction is that the difficult part is proving the implication (which is generally easier than a direct proof since you can assume something - namely the antecedent - is true which sets up showing that the consequent is true nicely).

§1.2 Complete Induction

There is an embodiment of regular induction known as **complete induction** (also referred to as strong induction) which can be useful in certain circumstances. The difference is in the inductive hypothesis during the inductive step of the proof. Usually, we just assume that $P(k)$ is true, but that also means we need to prove that $P(k+1)$ is true which is, of course, some work. Though, for some cases, it might be easier to frame the inductive step as follows:

Theorem 1.4 (Principle of Complete Induction)

Let P be some predicate. The principle of complete induction states that if

$$P(0) \text{ is true}$$

and

for any $k \in \mathbb{N}$, if $P(0), P(1), \ldots,$ and $P(k)$ are true then $P(k+1)$ is true

then

$$\forall n \in \mathbb{N}.P(n)$$

Essentially, we are framing the induction differently. Instead of just relying on the fact that $P(k)$ is true to prove that $P(k+1)$ is true, we utilize the fact that everything before $p(k)$ is true. So it is not a one-off chain reaction like with regular induction. Rather, we are saying that everything before $P(k)$ is true [3].

When to use complete induction vs. regular induction? First recall that the idea behind induction is to say that if "I can solve smaller versions of the problem" then I can "I can solve bigger versions of the problem". Use regular induction "when you know exactly how much smaller your 'smaller' problem instance is.". On the other hand, use complete induction when you are aware that things get smaller, but you are not exactly sure by how much[4] [3].

§1.3 Generalizing Induction

We are not limited to certain arbitrary contrainsts as it might seem so far. More specifically, we can use different step sizes greater than one if we'd like. Or, we can even use multiple base cases. More so, we can combine these two notions if we need to [5]. Another thing to note here is that the amount of base cases and the length of the step is often correlated. For example, if you have base cases $P(6), P(7)$ and $P(8)$, it makes sense to make the inductive step as $P(k+3)$–think about it, if you had a step length of one, then when you take the first base case, $P(6)$, you hit another base case ($P(7)$), and therefore, invalidate the induction. Furthermore, when you generalize induction and make these changes, your proof should still be of the same structure and length. You don't need to prove every part of the inductive step for each case–your goal is to generalize such a notion to each base case and beyond. Though, on the contrary, you must show that each base case is true.

§1.4 Summary

In sum, to prove something via induction, you should define some predicate P and do the following two things:

- Prove the base case to be true.

[3] I personally think trying to explain this in words is lacking. Watch the end of lecture [3] where Keith demonstrates an example with a row of people–it makes much more intuitive sense via that demonstration.

[4] This reminds of the idea behind when one should use a while loop (unaware in advance of how long something should repeat) or a for loop (aware of the exact number of iterations that need to be performed).

[5] Think as if you were programming a recursive function. You might have multiple base cases.

- Prove the inductive step (which is an implication). Assume some arbitrary value to be true, then prove that one step greater than that value is true. This allows you to conclude $\forall n \in \mathbb{N}.P(n)$.

§1.5 Examples

In this section, we will show various examples (with proofs) that illustrate induction. Examples will be taken from a listed source, and I will provide a short reflection on each if necessary. The main thing to takeaway from these is to realize how the principles listed above are applied in difficult problems. As such, when you go do your own problems, you have an intuitive sense and framework of what to do.

Example 1.5 (Source: [2])

Theorem: If exactly one coin in a group of 3^n coins is heavier than the rest, that coin can be found using only n weighings on a balance.

Proof: Let $P(n)$ be the following statement:

If exactly one coin in a group of 3^n coins is heavier than the rest, that coin can be found using only n weighings on a balance. We'll use induction to prove that $P(n)$ holds for every $n \in \mathbb{N}$, from which the theorem follows.

As our base case, we'll prove that $P(0)$ is true, meaning that if we have a set of $3^0 = 1$ coins with one coin heavier than the rest, we can find that coin with zero weighings. This is true because if we have just one coin, it's vacuously heavier than all the others, and no weighings are needed. For the inductive step, suppose $P(k)$ is true for some arbitrary $k \in \mathbb{N}$, so we can find the heavier of 3^k coins in k weighings. We'll prove $P(k+1)$: that we can find the heavier of 3^{k+1} coins in $k+1$ weighings.

Suppose we have 3^{k+1} coins with one heavier than the others. Split the coins into three groups of 3^k coins each. Weigh two of the groups against one another. If one group is heavier than the other, the coins in that group must contain the heavier coin. Otherwise, the heavier coin must be in the group we didn't put on the scale. Therefore, with one weighing, we can find a group of 3^k coins containing the heavy coin. We can then use k more weighings to find the heavy coin in that group. We've given a way to use $k+1$ weighings and find the heavy coin out of a group of 3^{k+1} coins. Thus $P(k+1)$ is true, completing the induction.

Notice the amount of words here as compared to the amount of math. This is ok–we explained our reasoning and intuition well which is what we needed to do. The biggest difference about induction compared to other proof techniques, in my opinion, is that induction requires a significant amount of thinking and intuition before you should attempt to write up a solution. Also notice that in the intuition paragraph, at the end, we mentioned that we just need to follow the same ideology for k more weighings. This is good style. Instead of explaining each step of the induction (like recursion), just show one step and say that it extrapolates. Furthermore, note that, for the inductive step, we *assumed* that such a case was true and what we then did was proved that a slightly bigger case $(k+1)$ was also true by building back down towards our smaller assumed case (in essense, we started with some bigger thing, than showed that, since we assumed it worked for the smaller thing, we can use said smaller thing to prove the bigger thing).

Example 1.6 (Source: [5])

Problem: Let $G_0 = 1, G_1 = 3, G_2 = 9$, and define

$$G_n = G_{n-1} + 3G_{n-2} + 3G_{n-3}$$

for $n \geq 3$. Show by induction that $G_n \leq 3^n$ for all $n \geq 0$.

Proof. Base Cases

$$\begin{aligned} n = 0 &: G_0 = 1 = 3^0 \\ n = 1 &: G_1 = 3 \leq 3^1 \\ n = 2 &: G_2 = 9 \leq 3^2 \end{aligned}$$

Inductive Step: Assume $n \geq 3$ and $P(k)$ for all k such that $0 \leq k \leq n$.

$$\begin{aligned} G_n &= G_{n-1} + 3G_{n-2} + 3G_{n-3} \\ &\leq 3^{n-1} + (3)3^{n-2} + (3)3^{n-3} \\ &= 3^{n-2}[3 + 3 + 1] \\ &= (7)3^{n-2} \\ &< (9)3^{n-2} \\ &= 3^n \end{aligned}$$

□

§2 Finite Automata

At the beginning of class, we asked ourselves a few questions in which one of those was:

"What are the limits of what we can do with a computer?".

This question defines the central question in **computability theory**. As we transition from pure discrete mathematics concepts to the theory of computation, we will apply such skills we learned in the first half. In the realm of computability theory, we will leverage mathematical proofs to form concrete arguments regarding various problems. The motivation behind finite automata (which we'll formally introduce below) is to allow us to leverage such mathematics in the first place. Moreover, consider the case where we have a problem that we want to know whether a computational solution exists or not. Presumably, such a problem might allow for an infinite (or a finite, but large) amount of inputs. We can't test them all. Even so, we can't identify, test, and come up with all possible solutions for certain problems (even with a quantum computer). So instead of working with the traditional computer system that we are all familiar with, to prove such broad and strong statements, we will work with **mathematical models** of them. More so, models allow use to make broad claims about computers since we argue that such models are just general models of computers. This means we can use them to formally prove things such as stating a problem is not solvable by any computer. And with that, we introduce to you automata[6].

Theorem 2.1 (Automaton)

An **automaton** (plural: automata) is a mathematical model of a computing device.

Such mathematical models look interesting and nothing like what you'd expect a "computer" to look like (see model below). First, we will give some characteristics of automata to get a broader sense of what they exactly are, then we will walk through how they work.

§2.1 Characteristics of Finite Automata

A **finite automaton** is a type of automata, with, well, a finite amount of resources.

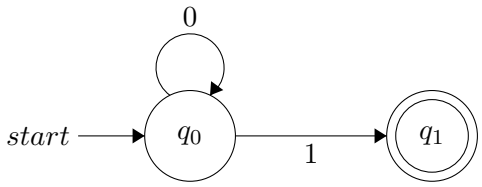
-

§2.2 How a Finite State Machine Works

At its heart, a computer is made up of 1's and 0's which represent transistor states on the hardware. A Finite State Machine (FSM) works in a similar manner. Consider the following FSM.

[6] I just want to enumerate (mostly for myself) a couple handy resources for drawing automata's in LaTeX:

- http://madebyevan.com/fsm/
- https://web.ma.utexas.edu/users/a.debray/lecture_notes/using_xy.pdf
- tikz package



- The FSM will take in, as input, a string filled with binary digits.
- Reading the string from left to right, each state (as represented by a q_x above) either directs, with arrows, where the next state will be. The first character in the string, starts at the state that the start arrow is pointing to.
- After following the directions of the FSM based on the input string, their will be a final state which represents whether the string is **accepted** or **declined**.

Acknowledgments

The LaTeXtemplate for these notes was obtained from Evan Chen's beautiful style file: evan.sty which is linked at: https://github.com/vEnhance/dotfiles/blob/main/texmf/tex/latex/evan/evan.sty. Furthermore, I'd also like to acknowledge Keith's wonderful lecture slides (available at the course website linked above) which are crafted with extraordinary detail. Indeed, some figures from those slides are displayed and cited in these notes.

Notes to self

Use examples for examples shown in lecture, theorem blue box type things for definitions (eventually learn how to make definitions go in the blue boxes then change them so that they say definitions). Additionally, define some proof template which uses the example box but either not red or just leave it red.

References

[1] **Indirect Proofs**, by Keith Schwarz. http://web.stanford.edu/class/archive/cs/cs103/cs103.1202/lectures/02/Slides02.pdf

[2] **Mathematical Induction, Part I**, by Keith Schwarz. http://web.stanford.edu/class/archive/cs/cs103/cs103.1202/lectures/12/Slides12.pdf

[3] **Mathematical Induction, Part II**, by Keith Schwarz. http://web.stanford.edu/class/archive/cs/cs103/cs103.1202/lectures/13/Slides13.pdf

[4] **How To Write Proofs: Mathematical Induction**, by Larry W. Cusick. http://zimmer.csufresno.edu/~larryc/proofs/proofs.mathinduction.html

[5] **6.042/18.062J Mathematics for Computer Science 2010, Midterm**, by Tom Leighton and Marten van Dijk. http://zimmer.csufresno.edu/~larryc/proofs/proofs.mathinduction.html