Overview of Induction

Chris Phifer

Fall 2020

Introduction

When we want to prove something about an infinite set (natural numbers, numerals, programs, terminating computations, derivations, and so on), the first tool we reach for is mathematical induction. Induction, although powerful, can be a bit gnarly: At bottom, we can find ourselves reasoning about implications about implications, sometimes about implications. This handout untangles the gnarl:

- It states, formally, the *principle of mathematical induction*.
- It illustrates the principle by way of analogy.
- It justifies the assumption of an inductive hypothesis.
- It discusses the situation where the inductive hypothesis is itself an implication.
- It introduces strong induction.
- It shows how to generalize induction from natural numbers to other forms of data.
- It give some concrete steps for writing an inductive proof.

Statement of the Principle

In its purest and most formal form, the induction principle may be scary. But if you know its vocabulary and notation, you'll start to understand how to use induction effectively and how to generalize it for use with more interesting data than natural numbers. On the natural numbers, the principle of induction is stated as follows:

The Induction Principle. *If $P(n)$ is a predicate about natural numbers, and if*

- *$P(0)$ is true, and*
- *for every natural number k, whenever $P(k)$ is true, $P(k+1)$ is also true,*

then for every natural number n, $P(n)$ is true.

An example $P(n)$ might be "there exists a decimal numeral that stands for n."

Supposing that we can establish the truth of the two bulleted statements, the induction principle lets us prove that some predicate P is true for every natural number. The hard part is the second bullet, which is where "inductive hypothesis" and "inductive step," come into play, and we spend the rest of this handout on it.

Demystifying Induction

An Analogy

Suppose you have an infinite number of dominoes;[1] you should be able to label each domino with a number starting from 0 and counting up by 1 (i.e., $0, 1, 2, \ldots$). You've conjured these dominoes in a straight line, each one standing on end. You park right at the beginning of this infinite line of dominoes, and, mischievous as you are, you knock the first domino over towards the next, and the chain begins to fall. Here's a question with a potentially obvious answer: supposing the dominoes are equally spaced and close enough that each one hits the next on the way down, will every domino fall?

The answer is "yes," and to prove it, we use the principle of induction. Let's tie the example to the formal statement of the principle: $P(n)$ stands for the predicate "domino n eventually falls over." We wish to prove that $P(n)$ holds for every domino. To apply the principle of induction, we must meet two proof obligations:

- We must show that $P(0)$ is true. That is, we must show that the first domino eventually falls over.
- We must show that for every natural number k, whenever $P(k)$ is true, $P(k+1)$ is also true. That is, we must show that if domino k eventually falls over, domino $k+1$ also eventually falls over.

[1] To be completely precise, that's a *countably* infinite number of dominoes.

The first proof obligation is easily met: We know that domino 0 falls over because we knocked it over ourselves. Now for the second obligation: Let k be a natural number, and suppose that $P(k)$ is true. That is, suppose that we know that domino k eventually falls; can we use this fact to show that $P(k+1)$ is true (i.e., that domino $k+1$ eventually falls)? Yes! As we noted, the dominoes are equally spaced such that each one hits is successor on the way down, and thus if we know that k eventually false, we also know that $k+1$ falls since k is guaranteed by its placement to knock over $k+1$.

Thus, because we've met both proof obligation for the principle of induction, $P(n)$ must be true for every natural number: Every domino must fall.

What still may be unclear is why—in the middle of that proof—it is legitimate to say "Let k be a natural number, and suppose that $P(k)$ is true." Doesn't it look like we're assuming what we're trying to prove? We're not. The induction principle demands proof of a particular *implication*, which has bearing on what assumptions can be made in constructing the proof.

Proving Implications

Consider a sentence of the form "if p, then q." How does one determine the truth of this entire sentence? In a logic or discrete math course, it is demonstrated that the truth values of a sentence like this can be established by considering the truth values of the subsentences, namely p and q; the words 'if' and 'then' merely act as a marker that we're invoking a particular truth function. Typically, we present these truth values in a table. Here is a table for the above sentence:

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	T

This tells us, for every possible combination of truth values that p and q may take on, what the truth value of "if p then q" will be. There are some things to note right away:

- The statement is always true if we start from a false premise (i.e. if p is false)
- When the premise is true, the truth of the implication is established precisely by determining whether the conclusion is true.

What these two observations mean in practice is that, in the context of a proof, it makes sense only to speak of cases in which the premise is taken as true - it simply isn't interesting to work under false premises, since any stated conclusion, no matter how absurd, results in a true implication.

The Induction Hypothesis

Here is the second proof obligation of the induction principle restated for convenience:

- for every natural number k, if $P(k)$ is true then $P(k+1)$ is true

This can be restated as an implication symbolically as:

- for every natural number k, $P(k) \to P(k+1)$

To prove this statement, then, by the previous discussion, *we assume that the statement P(k) holds* and use this hypothesis to establish that $P(k+1)$ is also true. The statement $P(k)$ is known as the *inductive hypothesis*; it is the assumed fact that's used to help establish the truth of the implication in question. In less formal terms, this *inductive step* establishes that if a smaller case works, then we can be sure that the next larger case works as well, exactly as with the dominoes, where if we knew that a domino somewhere along the way fell, we knew the next would necessarily fall as a consequence of being hit. There's no circularity here! We're simply tasked with showing that, in the case that the thing we want to show is true for some arbitrary case, then it is true for the 'next' case. This, paired with a proof that the smallest case (e.g. $P(0)$) holds, is enough to establish an arbitrarily long, deductively valid chain of reasoning beginning with the base case and applying the implication proved from the inductive hypothesis.

When the Induction Hypothesis is an Implication

In a programming-language proof, the inductive hypothesis is often an implication; for example, it might say that if an expression e evaluates, all its free variables are defined. To explore the structure of such a proof, we abstract, saying only that $P(k)$ has the form $A \to B$, where A and B are both propositions or judgments. For example, A might say that an expression evaluates, and B might say that all the expression's variables are defined. When $P(k)$ is written in the form $P(k) = A \to B$, there is still a k lurking in the

background, but it is assumed to appear somewhere inside A and B, not as an explicit parameter.

Since we'll be dealing with several implications, including both $P(k)$ and $P(k+1)$, it's helpful to have a way to name their parts: The two parts of an implication $A \to B$ are called the *antecedent* (A) and *consequent* (B). These terms are used in literature on logic.

The inductive proof of an implication has the same overall structure as any other inductive proof: we typically start with a base case $P(0) = A \to B$. Here our obligation is to prove a single implication. As discussed in the section on implications, proofs of implications proceed by assuming the antecedent A, and using this assumption and other facts to establish the truth of the consequent B. This form of mathematical proof is standard.

The complexity lies in the inductive step. In general, $P(k) = A \to B$ and $P(k+1) = C \to D$, and we are trying to prove $P(k) \to P(k+1)$, which is the same as proving $(A \to B) \to (C \to D)$. To prove this implication, we first assume the antecedent $A \to B$, as usual. Our obligation is now to prove the consequent $C \to D$, which is another implication! So in addition, we assume C. We now have the following assumptions:

1. $A \to B$, the antecedent of the outer implication
2. C, the antecedent of the consequent of the outer implication

Our goal is to prove D, the consequent of the consequent of the outer implication.

At this point, any good proof technique works, but programming-languages proofs tend to be stylized, and a good proof often takes three steps:

- Starting with assumption 2, use C to prove A.
- Apply assumption 1, using the A just proved to conclude B.
- Finally, use B to prove D.

As a shorthand for this technique, we can think of a chain of implications $C \to A \to B \to D$, where the middle implication is given to us,[2] and the first and last implications are what we have to prove. When this kind of inductive proof fails, it is almost always because one of the two implications ($C \to A$ or $B \to D$) is no good.

For an example of this technique in action, see the section "How to attempt a metatheoretic proof" in chapter 1 of *Programming Languages: Build, Prove, and Compare* by Norman Ramsey. This example uses an induction principle for a programming language's *operational semantics*, which we discuss briefly in a later section.

[2] It is the induction hypothesis.

Strong Induction

Another induction principle tends to be very useful in proof, inappropriately named the *strong principle of mathematical induction*. The name is inappropriate due to the fact that the principle is not stronger than 'normal' induction! It is, in fact, equivalent. The only difference in strong induction is in the inductive hypothesis: Rather than assuming that $P(k)$ alone is true, we assume that $P(0)$, $P(1)$, ..., $P(k)$ are true in order to prove $P(k+1)$. To continue the domino analogy, this would be useful if we knew that the dominoes were weighted, and the weight of *all* earlier dominoes in the line combined was required to knock the next domino over. Often, we find that it is necessary to rely on more information than just backing up a single step: Any time this is the case, strong induction is the way to go.

Some Generalizations

Other Inductive Structures

As already alluded to, the induction principle can be generalized very easily by simply replacing the appropriate parts of the bulleted proof obligations in the formal statement with the forms of the inductive data we wish to state a principle for. Here is a simple inductive definition of lists to demonstrate:

- [ ] is a list (this is an empty list)
- if x is an entity and xs is a list of entities of the same type as x, then $x :: xs$ is a list (where the double colon means "prepend the thing on the left to the list on the right.")

Perhaps it is already clear how to pattern match this inductive definition into a brand new induction principle that can be used to prove properties of lists. If not, here is a translation to demonstrate:

If $P(xs)$ is a predicate about lists such that:

- $P([\,])$ is true
- for every list ys, if $P(ys)$ is true then $P(y :: ys)$ is true,

then $P(xs)$ is true for every list.

Notice how similar this is to the original principle! In fact, if we look really closely, it becomes evident that *the principle for natural numbers is actually still being stated here*. Replacing every list with its length gives us back the original principle! In practice, this means to prove properties of lists, we can either induct on the form of the list as above (which is known as *structural induction*) or on the length of the list. Take some time to consider how the principle can be adapted for more interesting data, such as binary trees. First, write down the inductive definition of that data. Then, translate the induction principle on the naturals to use the forms of the data rather than numbers. Finally, try to connect back to the original principle by identifying an underlying natural number pattern as a sanity check.

Difficult Structures: Operational Semantics

While proving results about data is interesting and useful in showing that some code in a language is correct, how do we prove properties of *all* programs in a language? For this, we provide "metatheoretic" proofs, which is to say, "proofs about proofs"; these proofs proceed by inductive arguments *on the structure of derivation trees*. With the tools we've discussed, it is a good exercise to dissect this idea into a principled way of proving facts about all programs in a language.

A Process For Using Induction

With all of the details cleared up, here is a general process for constructing proofs by induction:

1. Identify that there is an inductive problem; that is, determine whether or not you're working with an infinity of finite, inductively defined structures
2. Determine the inductive hypothesis; note that sometimes you may need to strengthen the predicate in order to prove the result you want
3. Identify all of the cases, and begin to prove them. Some, but not all, will require that you use the inductive hypothesis
4. You're done!