Inductive Proofwriting Checklist

In Handout 26, the Guide to Inductive Proofs, we outlined a number of specific issues and concepts to be mindful about when writing inductive proofs. This proofwriting checklist distills down those concepts to smaller number of specific points that you should keep an eye out for when writing up your inductive proofs:

- ☐ ***Make $P(n)$ a predicate, not a number or function.***
- ☐ ***Watch your variable scoping in $P(n)$.***
- ☐ ***"Build up" if $P(n)$ is existentially-quantified; "build down" if it's universally-quantified.***
- ☐ ***Choose the simplest base case possible.***

The remainder of this handout goes into more detail about what each of these rules mean.

Make $P(n)$ a Predicate, Not a Number or Function

The principle of induction states that if you have a predicate P and the following are true:

- $P(0)$
- $\forall k \in \mathbb{N}.\ (P(k) \to P(k+1))$

then you can conclude that $\forall n \in \mathbb{N}.\ P(n)$ must be true.

It's important to note that P has to be a *predicate* for any of the above statements to be syntactically valid. Forgetting for the moment that we're dealing with induction, in First-Order Logic Land the statement $P(0)$ can only be true or false if P is a predicate, and the statement $P(k) \to P(k+1)$ only makes sense if $P(k)$ and $P(k+1)$ evaluate to truth values, which only happens when P is a predicate.

One of the most common mistakes we see in inductive proofs is to define P as something that isn't a predicate. For example, on the recurrence relations problem from Problem Set Five, if you want to prove that $a_n = 2^n$ for every $n \in \mathbb{N}$, you should ***not*** define $P(n)$ like this:

$$\text{⚠} \qquad P(n) = 2^n \qquad \text{⚠}$$

This doesn't work because P isn't a predicate; the quantity 2^n isn't something that evaluates to true or false. If we try using this P in the definition of mathematical induction given above, we'd say that if 2^0 is true and $\forall k \in \mathbb{N}.\ (2^k \to 2^{k+1})$ is true, then $\forall n \in \mathbb{N}.\ 2^n$ is true. The statement "2^0 is true" isn't mathematically meaningful, and we can't apply the $\to$ connective between the terms 2^k and 2^{k+1} because they're *quantities*, not *booleans*.

So before you submit your proofs, double-check that you've actually chosen P as a predicate. Just ask whether it's something that can be true or false (good!), or whether it's a quantity of some sort (bad!).

Watch Your Variable Scoping in $P(n)$

Another common mistake we see people make when defining the predicate P is to write something like this:

⚠ $P(n)$ is "for any $n \geq 1$, $|\mathbb{N}^n| = |\mathbb{N}|$." ⚠

Why exactly isn't this correct? After all, P is indeed a predicate: it either evaluates to true or false.

The issue here is one of *variable scoping*. To illustrate this, consider the following piece of C/C++/Java-esque code:

```
void doSomething(int n) {
    for (int n = 0; n < 137; n++) {
        // … do something with n …
    }
}
```

There's something weird about this code. This function takes in a parameter called `n`, which is supposed to be set by the person calling the function, but it then immediately proceeds to declare a new variable `n` inside the for loop. Depending on your programming language, this is either (1) really bad style or (2) a compile-time error.

The predicate P defined above makes the same mistake as this code. The problem is that there are two different variables n here: there's the variable n that's the argument of the predicate (kinda like the parameter n to the `doSomething` function), and then there's the variable n introduced by "for any n" (kinda like the variable n defined in the for loop). And just as the meaning of the code is either "uh, that's really weird" or "that's not even legal" (depending on which language you're using), mathematically the predicate P defined above is either "uh, that's really weird" or "that's not even legal" (depending on who you ask).

Treat the argument to the predicate P like an argument to a function. The caller specifies it, and you shouldn't say something like "for any n" or "for some n" inside of the definition of $P(n)$. Remember, induction ultimately lets you conclude that $P(n)$ is true for all $n \in \mathbb{N}$, and so the "for all $n \in \mathbb{N}$" part is provided *externally* to the predicate P.

"Build Up" if $P(n)$ is Existentially Quantified; "Build Down" if $P(n)$ is Universally-Quantified

Let's look at two of the inductive proofs we did in lecture: the proof that any square can be subdivided into n squares for any $n \geq 6$, and the proof that any tree with n nodes has exactly n-1 edges. If you look at the high-level structure of those proofs, you'll see that their inductive steps differ in a key specific way. In the proof that a square can be subdivided into smaller squares, the inductive step starts off with a subdivision of the square into k squares, then uses that to form a subdivision of the square into k+3 squares. In the proof that a tree with n nodes has n-1 edges, the inductive step starts with a tree of k+1 nodes, then breaks it apart into several smaller trees of at most k nodes each.

It seems like these proofs are going in opposite directions. In the square case, we start with a *smaller* object (size k) and grow it into a *larger* object (size k+3). In the tree case, we start with a *larger* object (size k+1) and shrink it into *smaller* objects (sizes at most k). What's going on here?

The answer has to do with how the predicate P is defined in each case. In the case of the square subdivision problem, then predicate P is (implicitly) *existentially-quantified*. That is, if we write it in a hybrid English/FOL notation, we'd get something like this:

$P(n)$ is "$\exists S$. (S is a subdivision of a square $\wedge$ S has n squares)"

In the case of the tree edges problem, the predicate P is (implicitly) *universally-quantified*. Written in a hybrid English/FOL notation, it looks like this:

$P(n)$ is "$\forall T$. (T is a tree with n nodes $\rightarrow$ T has n-1 edges)"

(Please don't actually write your predicates this way – it's just for expository purposes.)

Notice that in the square case, the predicate P is existentially-quantified, and in the tree case, the predicate P is universally-quantified. That's a really important distinction, and it explains why we work the way we do.

To see this, imagine we have some predicate P that looks like this:

$P(n)$ is "there is an X of size n where [...]"

If we want to prove that $P(n)$ is true for all $n \in \mathbb{N}$, then in our inductive step we'd need to prove that

> If
> there is an X of size k where [...],
> then
> there is an X' of size k+1 where [...].

Take a step back for a minute and think, abstractly, about how you prove something like this. We begin by assuming the antecedent. Since the antecedent is existentially-quantified, it means that we're assuming we have some honest-to-goodness object X lying around of size k that has some specific property. In other words, we can think of assuming the antecedent in this case as starting off with some concrete object in hand.

We then need to prove the consequent, which is an existential statement. This means that we need to somehow show that, somewhere out there, there's an object X' of size k+1 with some property. Given that we already have a handy object X lying around, it's very reasonable to see if we can start with that object X and then extend it into something larger.

In other words, we start with a *smaller* object (one of size k), then show how to convert it to a *bigger* object (one of size k+1).

On the other hand, if we have a predicate P that looks like this:

$P(n)$ is "for any object X of size n, X has property [...]"

then the inductive step would look like this:

> If
> for any X of size k, X has property [...],
> then
> for any X' of size k+1, X' has property […].

Think about how you prove a statement like this. We'd begin by assuming the antecedent. Since the antecedent is universally-quantified, we can think of it as a tool. It says "hey, as soon as you can actually find me an X of size k, I can step in and help you show that it has to some some key property." However, at the moment we don't actually have an object of size k lying around. Think of it as if we have a hammer rather than a nail – once we find a nail we can hammer it, but assuming the antecedent doesn't actually give us a nail to hit.

Next, we go to prove the consequent. The consequent is universally-quantified, and so if we want to prove it by a direct proof, we'd begin by choosing some object X' of size k+1, then trying to prove it has some property. So now we have an actual choice of object X' in hand. It's not quite the right shape for our antecedent (the antecedent only works when we get objects of size k, not k+1), so it's reasonable to try to see if we can produce some smaller object X out of our object X'. That way, we can use our antecedent to learn something new about X, which might in turn tell us something about X'.

In other words, we start with a *larger* object (one of size k+1) and then try to convert it to a *smaller* object (one of size k).

The difference between these two approaches is purely a difference in what quantifier is at the front of the predicate P. If it's existentially-quantified, we start with something smaller and then try to grow it. If it's universally-quantified, we start with something larger and shrink it down.

When you're writing up your proofs – ***and when you're still in the problem-solving stage*** – make sure that you keep these points in mind. Reread your proofs and check for the directionality in your induction – if you're going in a different direction than what we're proposing here, it likely means that there's a logic error in your proof.

Choose the Simplest Base Case Possible

All inductive proofs need to kick off the induction somewhere, and that's the job of the base case. Choosing the "right" base case is important to the proof, both in terms of correctness and in terms of proofwriting style. At the same time, choosing the right base case can be tricky, because inductive base cases often consider cases that are so small or degenerate that they bear little resemblance to the overall problem at hand.

Handout 26, the Guide to Inductive Proofs, goes into a bunch of detail about the importance and value of choosing the right base case, and in the interest of space we won't repeat all of those details here. What we can do, though, is provide a series of questions you should think through to make sure that you've picked your base cases intelligently:

1. ***Are all the numbers you need to have covered covered?*** For example, if you're going to prove that something is true for all natural numbers, you'll need to ensure that you've covered the zero case, so you'd probably want to pick a base case like $n = 0$. If you start later than this, it's likely a logic error.
2. ***Have you thought through degenerate cases?*** Often times, the base case of an inductive proof involves an extreme sort of edge case (a set of no elements, an implication that's vacuously true, a sum of no numbers, a graph with no nodes, etc.) It can feel really, really weird working with cases like these the first time that you're exposed to them, but it gets a lot easier with practice. If you're uncertain whether the result is true in some extreme case, it's best to ask the course staff for input. You'll build a good intuition for what works and what doesn't over time.
3. ***Does the logic in your inductive step apply to your base case?*** Once you've picked your base case, look at the reasoning in it. Is the reasoning there just a special case of the reasoning in your inductive step? If so, that might indicate that you haven't actually picked the simplest possible base case and that you might be able to pick an even simpler base case.
4. ***If you have multiple base cases, are they truly necessary?*** Some proofs by induction really do need multiple base cases, such as the square subdivision problem from lecture (where we need one base case for each remainder modulo three). However, most of the time we see proofs that have multiple base cases in them, we find that at least one of them isn't necessary. Often times, that extra base case can be eliminated by asking question (3) from this list.