Choose Your Own Induction Principle

CPSC 509: Programming Language Principles

Ronald Garcia*

05 March 2014

So far in the course we have made heavy use of a particular set of reasoning principles. We started with basic logic and set-theoretic reasoning, and then added a trinity of tools to our arsenal: inductive definitions of sets, proofs of properties by induction, and functions defined using recursive equations. The latter three tools were all introduced as coming straight out of our inductive definitions: the structure of our definitions determined the structure of our reasoning principles.

Now I assume that this class is not the first time that you've heard of induction, and I hope that so far I have been able to show you how the proofs that you read in papers and textbooks can be mapped back to a much more precise and rigorous formulation based on a property and a principle of induction (rather than some blather about "base cases" and "induction hypotheses"). But there is still a bit of disconnect left. In all likelihood you've seen arguments that appeal to "induction on the number n" where n may be the length of a sequence, the size of a matrix, or maybe the number of interesting elements left in a set.

The purpose of this lesson is to tie up this loose end. The key idea is that proofs by induction like the ones described above are just particular instances of what we've been doing in this class. Hopefully you'll see in this section why I have been so persnickety about the nature of inductive definitions. My hope is to empower you to use our three tools with comfort and fluency, *in their fullest generality*, because by goodness sometimes you need need that kind of power to solve a problem!

The key takeaway point of this note will be that *a set can have many different induction principles, and which one you choose can affect how you reason about your set.*

Given a set that you've defined, you might find that proving a property of that set or defining a function on that set may require some ingenuity in picking your induction principle. Before we can pick new ones, we need to know how to *create* them.

1 Inductive Definitions, Revisited

To make sure that you understand the subtleties of this section, in a way that you can carry with you beyond this class, let's first quickly review some topics from our second lecture of the course. We'll start with where we are now, and then work backwards to where we started from.

Consider the following BNF for a tiny language

$$\begin{array}{rcl} & & b \in B \\ b & ::= & \mathtt{t} \mid \mathtt{f} \mid b \bullet b \end{array}$$

Stepping backwards, this is just a concise shorthand for an inductive definition: Given our assumption $r \in \textsc{Tree}$, we define the set $B \subseteq \textsc{Tree}$ using the following rules:

$$\frac{}{\mathtt{t} \in B}\ (\mathsf{t}) \qquad \frac{}{\mathtt{f} \in B}\ (\mathsf{f}) \qquad \frac{r_1 \in B \quad r_2 \in B}{r_1 \bullet r_2 \in B}\ (\bullet)$$

Now remember: the $\in B$ part of the inductive definitions is purely commentary: it's syntactic sugar to make the definitions easier to read. Our rules are *really* just as follows:

$$\frac{}{\mathsf{t}}\ (\mathrm{t}) \qquad \frac{}{\mathsf{f}}\ (\mathrm{f}) \qquad \frac{r_1 \quad r_2}{r_1 \bullet r_2}\ (\bullet)$$

Now, recall that each of these rules stands for some set of *rule instances*, where any metavariables are replaced with every possible element that fits and satisfies the side-conditions. To make this concrete, here are two different instances of the rule (•):

$$\frac{\mathsf{t} \quad \mathsf{f}}{\mathsf{t} \bullet \mathsf{f}}\ (\bullet) \qquad \frac{\mathsf{Ringo} \quad \mathsf{Starr}}{\mathsf{Ringo} \bullet \mathsf{Starr}}\ (\bullet)$$

The first rule instance is appealing because it can appear in a full derivation:

$$\mathcal{D} = \frac{\dfrac{}{\mathsf{t}}\ (\mathrm{t}) \quad \dfrac{}{\mathsf{f}}\ (\mathrm{f})}{\mathsf{t} \bullet \mathsf{f}}\ (\bullet)$$

but the second instance, which is also a perfectly fine instance, will never appear within any actual derivation of $r \in B$.

Another way to think about it that may help is that a rule stands for a set of rule instances:

$$\begin{aligned}(t) &= \{\ \frac{}{\mathsf{t}}\ \} \\ (f) &= \{\ \frac{}{\mathsf{f}}\ \} \\ (\bullet) &= \left\{ \frac{r_1 \quad r_2}{r_1 \bullet r_2} \;\middle|\; r_1, r_2 \in \textsc{Tree} \right\}\end{aligned}$$

Peeling back the curtain just a little more[1], let me fess up that each rule instance is really just a set of premises and a conclusion paired together. For example,

$$\frac{\mathsf{a} \quad \mathsf{b}}{\mathsf{a} \bullet \mathsf{b}}\ (\bullet) \equiv \langle \{\, \mathsf{a}, \mathsf{b} \,\}, \mathsf{a} \bullet \mathsf{b} \rangle$$

So really

$$(\bullet) = \{\ \langle s, r \rangle \in \mathcal{P}(\textsc{Tree}) \times \textsc{Tree} \mid s = \{\ r_1, r_2 \ \} \wedge r = r_1 \bullet r_2 \ \}$$

Aren't you glad we avoided this rabbit hole earlier? Now with this realization, we observe that our full set of rule instances is $\mathcal{R} = (t) \cup (f) \cup (\bullet)$. We'll call the set of derivations that can be constructed from this set of rule instances $\textsc{Deriv}[\mathcal{R}]$.

Finally, our set B is defined by filtering $\textsc{Tree}$ down to only those members that have derivations from instances of our rules:

$$B = \{\ r \in \textsc{Tree} \mid \exists \mathcal{D} \in \textsc{Deriv}[\mathcal{R}].\mathcal{D} :: r \ \}$$

Notice that I've left out the syntactic sugar, to keep things precise.

1.1 Defining B yet again!

This subsection is meant to bring home a subtle concept that you may not have picked up before.

Throughout the course, we've been defining sets in terms of other sets. Sure there are a few that we assume from time to time, like pairs and TREEs, but that's just to keep things abstract and not ridiculously low-level. Nonetheless, so long as we have sets, we can define others. In particular, so long as we have a set of rule instances, we can use it to inductively define a set. Think back to Homework 2 where we "inductively" defined a pretty ridiculous set with a finite number of elements. Induction doesn't care if you create an infinite or finite set. In fact, you can only inductively define an infinite set by starting with some other infinite set (e.g. $\textsc{Tree}$ was already infinite before we ever defined B). Induction just happens to

[1] ideally without confusing you...

be a very effective tool for *carving* out a desirable infinite set from some other (already) infinite set. But you can carve finite sets out of finite sets too. Sometimes doing so can be convenient: it's analogous to using a loop to write some code that repeats, say, 70 times (inductive definition) rather than repeating yourself painfully 70 times (extensional definition).

Anyway, consider the following new set of inductive rules.

$$\frac{}{\mathsf{t}}\ (\mathrm{t2}) \qquad \frac{}{\mathsf{f}}\ (\mathrm{f2}) \qquad \frac{b_1 \quad b_2}{b_1 \bullet b_2}\ (\bullet 2)$$

The only change from the last section is that the r's have been replaced with b's. In the past I've said that you can't define $b \in B$ this way because we don't know what b is yet, so that would be circular. In this case though, we defined B in the last subsection, so this defines *some other* set, which we'll call C, in terms of the pre-existing set B. So....what's going on?

Well, when it comes to this inductive definition, some things change and some things stay the same. First, be aware that $C = B$: they are *exactly the same set*. Why?

Because if we take $\mathcal{R}_2 = (t2) \cup (f2) \cup (\bullet 2)$ Then it's a fact that for every $b \in B$, there's a derivation $\mathcal{D} \in \textsc{Deriv}[\mathcal{R}_2]$ such that $\mathcal{D} :: b$, so $B \subseteq C$. And furthermore, the rules $\mathcal{R}_2$ ensure that $C \subseteq B$. Thus,

$$C = \{\, b \in B \mid \exists \mathcal{D} \in \textsc{Deriv}[\mathcal{R}_2].\mathcal{D} :: b \,\} = B.$$

A side-note though. Notice that every instance of $(\bullet 2)$ is also an instance of $(\bullet)$ but not vice-versa (remember Ringo • Starr?), so $(\bullet 2) \subsetneq (\bullet)$). So these are not the *same* inductive definitions, but they do define the same set (B), inductively!

Question: Why the heck would I want to do that? Well, this particular case was not especially interesting, but there is one small change. First, we're going to drop the name C, because $C = B$, and there's no need for a different name. But look at the principle of induction on derivations:

Proposition 1. *Let P be a property of derivations $D :: b$, then*

$$\begin{aligned}
&P(\tfrac{}{\mathsf{t}}) \\
\wedge\ &P(\tfrac{}{\mathsf{f}}) \\
\wedge\ &\left(\forall b_1, b_2 \in B.\, \forall D_1, D_2 \in \textsc{Deriv}[\mathcal{R}_2].\, \mathcal{D}_1 :: b_1 \wedge \mathcal{D}_2 :: b_2 \wedge P(\mathcal{D}_1) \wedge P(\mathcal{D}_2) \implies P\left(\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \hline b_1 \bullet b_2 \end{array}\right)\right) \\
\implies &\forall D \in \textsc{Deriv}[\mathcal{R}_2].P(D).
\end{aligned}$$

Notice how there are now *no* references to $r \in \textsc{Tree}$? It's *all* in terms of $b \in B$ now. You can think of this as giving the induction principle a nice "trim around the ears" to get rid of some details we don't care about, artifacts of our principle of induction. *This* is the principle you are more likely to see in other textbooks, but we've come through it via a more foundational approach.

In the following sections, we will go farther. We will produce new inductive definitions for old sets where the rules and rule instances are *quite different* from the old ones (maybe even a little crazy!). As a result, we will get principles of induction that themselves are quite different. This difference can be really helpful, sometimes critical.

2 Example: Induction in Arithmetic

To motivate this idea, we turn briefly to arithmetic. In all likelihood, you've come across induction in other classes, but it was all about numbers. In fact, the explicit principle of induction is often left unstated: you're just supposed to understand the idea of "proof by induction". Let's make explicit the principle that you were using in those classes.

Proposition 2 (Principle of Mathematical Induction)**.** *Let $P(n)$ be a property of natural numbers. Then P holds for all natural numbers if:*

1. *$P(0)$ holds, and*
2. *If $P(n)$ holds then $P(n+1)$ holds.*

Here is a simple proof of a mathematical property by induction. You may have seen this before.

Proposition 3 (Summation Theorem)**.** *For all numbers n, the sum of all numbers from 0 to n is $(\Sigma_{i=0}^{n}\ i) = \frac{n^2+n}{2}$.*

Proof. By mathematical induction

Case (1.). Then the sum of numbers from 0 to 0 is 0, which is the same as $\frac{0^2+0}{2}$.

Case (2.). Let n be some particular number, and suppose $P(n)$, that is, suppose $(\Sigma_{i=0}^{n}\ i) = \frac{n^2+n}{2}$. We now want to prove that $\left(\Sigma_{i=0}^{(n+1)}\ i\right) = \frac{(n+1)^2+(n+1)}{2}$. Right away, let's observe that

$$\frac{(n+1)^2+(n+1)}{2} = \frac{(n^2+2n+1)+n+1}{2} = \frac{n^2+3n+2}{2}.$$

Next, let's observe that

$$\left(\Sigma_{i=0}^{(n+1)}\ i\right) = (\Sigma_{i=0}^{n}\ i) + (n+1) = \frac{n^2+n}{2} + (n+1)$$

by the definition of summation Σ and then by our assumption (i.e. the induction hypothesis). Then, with some basic arithmetic, we get

$$\frac{n^2+n}{2} + (n+1) = \frac{n^2+n+2(n+1)}{2} = \frac{n^2+n+2n+2}{2} = \frac{n^2+3n+2}{2},$$

And we're done!

□

That wasn't too bad compared to the proofs we've been doing so far.

Now if we look back at the principle of induction, it seems like it corresponds to the following "definition" of natural numbers:

$$\frac{}{0 \in \mathbb{N}} \qquad\qquad \frac{n \in \mathbb{N}}{n+1 \in \mathbb{N}}$$

Notice that here we're defining natural numbers in terms of exactly the set of natural numbers![2] That's perfectly fine because $\mathbb{N} \subseteq \mathbb{N}$.

You may also have heard of proving properties of numbers by Strong Induction. We'll state its principle here explicitly.

Proposition 4 (Principle of Strong Mathematical Induction)**.** *Let $P(n)$ be a property of natural numbers. Then P holds for all natural numbers if:*

1. *For all numbers n, if $P(n')$ holds for all $n' < n$ then $P(n)$.*

This principle of induction looks quite different from the ones that we've seen before. There is nothing that looks like a "base case", and in fact there's only one case!

Here's an example of a classic proof that can be proved nicely by strong induction:[3]

Proposition 5 (Division Theorem)**.** *For all natural numbers n, and natural numbers $m > 0$, there are natural numbers q and r such that $n = qm + r$ and $r < m$.*

This should not be news to you: n divided by m is q with remainder r!

[2] Let's not worry about how the natural numbers were originally defined!

[3] I swear, we'll stop with the maths soon!

Proof. By strong induction on n. The induction as stated doesn't impose cases, so we can dive right in.

Suppose n is a natural number and the desired property holds for all $i < n$. Well, we can decompose this problem into two cases, depending on the relationship between n and m. Either n is less than m or it's greater-or-equal to m.

Case ($n < m$). Let $q = 0$ and $r = n$. Then $n = qm + r$ and we know by assumption that $r < m$, so this case is done.

Case ($n \geq m$). Since $n \geq m$, we know that $n - m$ is a natural number. Furthermore, since $m > 0$, we know that $n - m < n$, so by the induction hypothesis, $n - m = q_2 m + r_2$ for some q_2 and $r_2 < m$. Then by adding m to both sides, we get:

$$\begin{aligned} n &= q_2 m + r_2 + m \\ &= (q_2 + 1)m + r_2. \end{aligned}$$

Then, if we let $q = q_2 + 1$ and $r = r_2$, we know that $n = qm + r$ and $r = r_2 < m$. This case is done!

□

There are a few interesting things worth noticing in this proof. First, the principle of strong induction has only one case, so the proof uses it, but internally the problem is broken into two cases ($n < m$ and $n \geq m$) which it turns out to be nice to handle separately. So *the case structure of the proof is independent of the induction principle*. To date, our cases have always been dictated by the induction principle, though sometimes we still break the problem down more. For example, in a proof about big-step evaluation by induction on TERMs, when faced with an if expression, we consider separately whether the predicate evaluates to true and whether it evaluates to false. Here we are given *one big fat case* and we can break it down as we please. This flexibility comes in handy some times, just like above.

Another thing worth noting. Throughout the course I have emphasized that our proofs often correspond to programs (interpreters, steppers, etc.). What about the proof of the above mathematical statement? If we write the statement formally:

$$\forall n, m \in \mathbb{N}.\, m > 0 \implies \exists q, r \in \mathbb{N}.\, n = qm + r \wedge r < m. \tag{1}$$

We can see that it roughly says "give me two natural numbers n, and m, where m is positive, and I will give you back two numbers q, r that are the quotient and remainder of n divided by m.

Indeed, the computational content of the above proof is *exactly* integer division! And indeed it is a recursive function. Can you code it up?

Back to induction. Now, if we think about it, if we try to map the principle of strong induction back to induction by definitions, we get a single rule:

$$\frac{0 \in \mathbb{N} \quad 1 \in \mathbb{N} \quad \ldots \quad n-1 \in \mathbb{N}}{n \in \mathbb{N}}$$

Or slightly more formally

$$\frac{\{\, n' \in \mathbb{N}' \mid n' < n \,\}}{n \in \mathbb{N}'}$$

And instances of this rule are as follows:

$$\frac{}{0 \in \mathbb{N}} \qquad \frac{0 \in \mathbb{N}}{1 \in \mathbb{N}} \qquad \frac{0 \in \mathbb{N} \quad 1 \in \mathbb{N}}{2 \in \mathbb{N}} \qquad \frac{0 \in \mathbb{N} \quad 1 \in \mathbb{N} \quad 2 \in \mathbb{N}}{3 \in \mathbb{N}} \qquad \ldots$$

Here our rule stands for instances with *any number* of premises, not just some fixed number, as our old language-based rules suggested. This is fine so long as we understand what counts as a legal rule instance, and we do. This is exactly our description of rule-instances as pairs. Our rule instances for strong mathematical induction are

$$\mathcal{R}_{\text{strong}} = \{\, \langle s, n \rangle \in \mathcal{P}(\mathbb{N}) \times \mathbb{N} \mid s = \{\, n' \in \mathbb{N} \mid n' < n \,\} \,\}$$

For some simplicity, let's give the premise set a name based on the less-than comparator.

$$<\mathbf{n} = \{\, n' \in \mathbb{N}' \mid n' < n \,\}$$

Then our rule instance is:

$$\frac{<\mathbf{n}}{n}$$

Notice two things: First, there is a rule instance for every $n \in \mathbb{N}$ by definition. Second, there is a derivation for every $n \in \mathbb{N}$ that can be built up using the derivations for all of the smaller n's. In fact, the height of the derivation tree is bounded by the size of the number n itself. Thus, the above rule definitely defines the same set $\mathbb{N}$.

The takeaway lesson here is that the less-than operator $<$ can be used to define the sets $<\mathbf{n}$ of numbers less-than-n, which in turn gives us another inductive definition of numbers and thereby gives us a principle of induction.

Furthermore, the inductive rules here are definitely deterministic, since for each number n, there is only one rule instance, the one that takes in all the numbers less than it. Thus, we could define a function by introducing a corresponding principle of recursion based on strong induction, called *course of values recursion*. In fact the division function I described above, the computational content of the proof of the division theorem, is defined by course of values induction.[4]

3 Generalizing Strong Induction: Well-founded Induction

What's fascinating and powerful is that we can come up with new "less-than" operators that give us even more new induction principles. Rather than constantly developing new sets of inductive rules, and carefully showing that they will define the same set at we started with, we instead just define a new binary relation on our set, and use *that* relation to properly construct the rules and give us a new reasoning principle. First we start with an example and then we state the general criteria for doing this.

Let's consider the following relation on natural numbers:

$$n <_1 m \quad \text{iff} \quad m = n + 1.$$

This looks a lot like the $<$ we know and love, but not quite. For instance, $5 <_1 6$, but it's not true that $5 <_1 7$. In short, it only relates adjacent natural numbers.

Now consider the inductive rule:

$$\frac{<_1\mathbf{n}}{n}$$

where $<_1\mathbf{n}$ has the analogous meaning to $<\mathbf{n}$. This single rule induces the same set of rule instances as we got for mathematical induction! The result is a different phrasing of the same principle of mathematical induction:

Proposition 6 (Principle of Mathematical Induction)**.** *Let $P(n)$ be a property of natural numbers. Then P holds for all natural numbers if:*

1. *For all numbers n, if $P(n')$ holds for all $n' <_1 n$ then $P(n)$.*

Funny, that doesn't *look* the same! For example, there's only one case, and no base case! Nonetheless, this is the same inductive principle, and if we choose, we can break our problem up along the cases we had before. Here is a rephrasing of the general structure of our earlier Summation Theorem proof:

Proof. By mathematical induction.

Suppose n is a number and the statement holds for all $i <_1 n$. Well, we can decompose this problem into two cases, depending on what kind of natural number n is:

Case ($n = 0$). Then the sum of numbers from 0 to 0 is 0, which is the same as $\frac{0^2+0}{2}$.

Case ($n = m + 1$ for some $m \in \mathbb{N}$). Then by our assumption (i.e. the induction hypothesis), $P(i)$, that is ...

□

The proof is basically the same, but we got to choose how to break the problem space up. It turns out that for the $n = 0$ case, we can't rely on the induction hypothesis, because *there are no numbers $n <_1 0$*, but that's okay because we didn't need any.

[4] See the Wikipedia page on *course of values recursion* for more details, but ignore the Fibonacci function, because course of values recursion is overkill for it (see the next footnote)!

3.1 Well-founded relations

So we've been able to define two induction principles by establishing a "less-than" relation, but does this work more generally? The answer is *yes*, and the kind of relation that we want for this is of the following kind.[5]

Definition 1 (Well-foundedness)**.** *A binary relation $\sqsubset \subseteq S \times S$ is* well-founded *if it has no infinite descending chains, i.e. no unbounded strings of the form:*

$$\cdots \sqsubset s_3 \sqsubset s_2 \sqsubset s_1 \sqsubset s_0$$

As it turns out, given any set S and a well-founded relation $\sqsubset$, we immediately get a corresponding principle of induction. This makes for a very general statement:

Proposition 7 (Principle of Well-founded Induction)**.** *Let S be a set, let $\sqsubset$ be a well-founded relation on S, and let $P(s)$ be a relation on S. Then $P(s)$ holds for all $s \in S$ if*

1. *For all elements $s \in S$, if $P(s')$ holds for all $s' \sqsubset s$ then $P(s)$.*

Formally:

$$(\forall s \in S.\, (\forall s' \in S. s' \sqsubset s \implies P(s')) \implies P(s)) \implies \forall s \in S.\, P(s) \tag{2}$$

That's quite a mouthful! But a powerful mouthful it is.

3.2 Common Examples

Now that we have a general framework for defining induction principles, we can build up a common set of principles that come up a lot in practice. This way, you can quickly and easily tell what induction principle is at work in any particular function definition. I'm often surprised how many textbooks and papers claim that a particular definition is justified by some particular recursion principle when in fact it's not.[6]

Coordinate Induction First, lets tackle the justification for the $append$ function.

Definition 2. *Suppose A and B are sets, and A has a well-founded relation $<_A$. Then we can define another well-founded relation $\sqsubset$ on pairs from $A \times B$ such that $\langle a_1, b_1\rangle \sqsubset \langle a_2, b_2\rangle$ if and only if $a_1 <_A a_2$.*

This well-foundedness relation is completely determined by the first elements in every pair. If B has a well founded relation, then we could instead produce a foundation relation based on the second member of each pair. Such a relation corresponds to the *coordinate foundation*-based recursion principle used to define $append$. This generalizes to any ordered sequence of sets where one of them is well-founded.

Coordinate induction is particularly useful when you have to deal with tuples where only one component is treated inductively. This happens in proofs over tuples, where the extra arguments are basically "parameters". The same happens when you wish to define a function of multiple arguments, but the definedness of the function is determined "by recursion on the structure of the nth element", and the rest just come along for the ride.

Simultaneous Induction Some situations look like the above, where you have $A \times B$ but you want to consider well-foundedness relations on *both* coordinates at the same time, in lock-step. We can do that. In particular, we can define *simultaneous foundation*, which relies on every coordinate of a tuple decreasing: $\langle a_1, b_1\rangle \sqsubset \langle a_2, b_2\rangle$ if and only if *both* $a_1 <_A a_2$ and $b_1 <_B b_2$.

[5] In fact, the relation that I would argue is a crisper basis for the Fibonacci function is: $n <_{\text{fib}} m$ iff $m = n + 1 \vee m = n + 2$. Work through the details to see why.

[6] To be fair, in most cases what's presented is indeed a valid definition, but only according to some other recursion principle.

Lexicographic Induction In both coordinate induction and simultaneous induction, it's quite clear that the resulting relation is well-founded, because something that can only decrease so much decreases each time. Less obvious, but very useful, is *lexicographic foundation*, which we define for the simple case of pairs

Definition 3. *Suppose A and B are sets with respective well-founded relations $<_A$ and $<_B$. Then we define the lexicographic foundation relation as follows: $\langle a_1, b_1\rangle \sqsubset \langle a_2, b_2\rangle$ iff*

1. *$a_1 = a_2$ and $b_1 <_B b_2$ or*
2. *$a_1 <_A a_2$.*

Here are some examples of this well-founded relation, taking both A and B to be natural numbers:

1. $\langle 4, 6\rangle \sqsubset \langle 5, 7\rangle$
2. $\langle 5, 6\rangle \sqsubset \langle 5, 7\rangle$
3. $\langle 4, 3000\rangle \sqsubset \langle 5, 7\rangle$

In the lexicographic foundation relation, the second coordinate must go down if the first coordinate stays the same, but if the first coordinate goes down, then the second coordinate may do whatever it likes. Lexicographic induction corresponds to performing multiple inductions nested inside one another. Often doing them all as one lexicographic induction will be more concise, and possibly even more clear.

Measure Induction Some of the most common and useful inductions are built around some function from a set of interest to the natural numbers, which we will call a *measure function*.[7] For example, in class we defined a function $size : \textsc{Term} \to \mathbb{N}$ that classifies the size of a term. Given such a function, it's easy to define a well-ordering on terms:

$$t_1 \sqsubset t_2 \text{ if and only if } size(t_1) < size(t_2).$$

It's easy to see that this is a well-founded relation because every chain corresponds to a descending chain of natural numbers. Using this relation gets us what would be called "induction on the size of Terms t."

Proposition 8 (Principle of Induction on $size(t)$)**.** *Let $P(t)$ be a property of* Terms*. Then $P(t)$ holds for all terms t if the following is true: If $P(t')$ holds for all terms t' such that $size(t') < size(t)$ then $P(t)$ holds as well.*

You can establish similar principles for many number-valued functions: the *depth* of a term t, the *length* of a reduction sequence $t \longrightarrow^* t$, and even the *height* of a derivation tree $\mathcal{D}$.

Reduction Induction Here is what I consider the strangest, but terribly useful, induction I have ever had to use. I am putting it here just to document it and to maybe give you some crazy ideas. I had a language in which program reduction $t \longrightarrow t$ did not necessarily terminate, but if you limited yourself to a certain *subset* of reductions $t \longrightarrow^S t$, those would always terminate. I defined a well-founded relation as $t < t'$ if and only if $t' \longrightarrow^S t$. Since $\longrightarrow^S$ terminates (I proved that), it follows that $<$ is well-founded, and with it came a useful induction principle. Doing this helped me prove something I otherwise did not know how to prove.

4 Aside: Mutually Inductive Definitions

In addition to covering the idea of well-founded induction, let's take a little aside to talk *even more* about inductive definitions and their subtleties.

[7] I just made this name up so don't necessarily expect to find it in the literature, but it makes so much sense, that it wouldn't surprise me if someone else came up with it before, or if I saw it elsewhere and just stole it.

4.1 Preconditions vs. Side Conditions

To explain those, let's consider a simple example that you've already seen. Suppose a set of terms $t \in \text{TERM}$ and a small-step relation on it $\longrightarrow \subseteq \text{TERM} \times \text{TERM}$. Then we can take the reflexive-transitive closure of this relation: Here are the rules that define $\longrightarrow^*$:

$$\text{(incl)}\frac{t_1 \longrightarrow t_2}{t_1 \longrightarrow^* t_2} \qquad \text{(refl)}\frac{}{t \longrightarrow^* t} \qquad \text{(trans)}\frac{t_1 \longrightarrow^* t_2 \quad t_2 \longrightarrow^* t_3}{t_1 \longrightarrow^* t_3}$$

The (refl) and (trans) rules are the usual thing we are used to, but notice that the (incl) rule has a use of $\longrightarrow$ on top. To *really* get what's going on, let's desugar these rules for $\longrightarrow^*$:

$$\text{(incl)}\frac{}{\langle t_1, t_2\rangle} \quad \text{where } t_1 \longrightarrow t_2 \qquad \text{(refl)}\frac{}{\langle t, t\rangle} \qquad \text{(trans)}\frac{\langle t_1, t_2\rangle \quad \langle t_2, t_3\rangle}{\langle t_1, t_3\rangle}$$

In the desugared form, it becomes apparent that the top of the sugared (incl) rule is really a *side-condition*, not a premise. Premises can only be things that *might* be elements of the set being presently defined. Of course, if two terms t_1 and t_2 satisfy the side condition $t_1 \to t_2$, then by definition, there must be a derivation of this fact (assuming that $\longrightarrow$ was also inductively defined), but that derivation is independent of the current derivation.

Now, as a side note, consider the following seemingly peculiar inductive definition:

$$\text{(incl)}\frac{t_1 \not\longrightarrow t_2}{t_1 \☠ t_2} \qquad \text{(refl)}\frac{}{t \☠ t} \qquad \text{(trans)}\frac{t_1 \☠ t_2 \quad t_2 \☠ t_3}{t_1 \☠ t_3}$$

Is this definition of $\☠ \subseteq \text{TERM} \times \text{TERM}$ alright? Sure! the natural definition of $\not\longrightarrow$ is as follows:

$$\not\longrightarrow = \{\, p \in \text{TERM} \times \text{TERM} \mid p \notin \longrightarrow \,\}$$

Well that's a well-defined set. And once again its use on top of (incl) is a side-condition, so it produces a perfectly fine set of rule instances. So what we get is the reflexive-transitive closure of "doesn't single-step to". Why would I want that? *Hell if I know*, but that's not the point: the point is that it's a well-defined set!

Why emphasize this? Because suppose I wrote the following and claimed that it was an inductive definition:

$$\text{(incl)}\frac{t_1 \longrightarrow t_2}{t_1 \☹ t_2} \qquad \text{(refl)}\frac{}{t \☹ t} \qquad \text{(trans)}\frac{t_1 \☹ t_2 \quad t_2 \not\☹ t_3}{t_1 \☹ t_3}$$

Well now *this* is problematic. If you try to desugar this, then you get stuck trying to translate the $t_2 \not\☹ t_3$. It has no meaning yet because you haven't defined $\☹$ yet, let alone $\not\☹$.

As a result of this phenomenon, many have come to learn the (correct) belief that having "negative premises" is fundamentally wrong, but unfortunately then confuse negated side-conditions with negated preconditions, a behaviour that is somewhat understandable of reasonable since side-conditions often get typeset as if they were preconditions. However, the result is that some perfectly fine inductive definitions get incorrectly criticized.

4.2 Mutually Inductive Definitions

So far, you've only seen inductive definitions where all of the preconditions are exactly references to the elements of the set being defined. There's one circumstance that seems like it's not like this: when two or more sets are mutually defined inductively. This is different from the example above where one set is defined and then used as a side-condition to define the next.

We proceed with an example. Suppose a programming language that is kind of like our language of mutable references, but it distinguishes terms that are executed for their side-effect only (which we'll call *statements*) from terms that are executed for their values (which we'll call expressions). However, we will

allow expressions to execute statements before producing a value, and we will allow statements to evaluate expressions. Here we'll just introduce the syntax, and I'll leave it to you to come up with a semantics.

$$n \in \mathbb{Z}, \quad x \in \text{VAR}, \quad e \in \text{EXPR}, \quad s \in \text{STMT}$$
$$\begin{array}{rcl} e & ::= & n \mid \mathsf{true} \mid \mathsf{false} \mid x \mid e + e \mid e\text{-}e \mid e = e \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid \mathsf{do}\ s\ \mathsf{in}\ e \\ s & ::= & x := e \mid s;s \mid \mathsf{if}\ e\ \mathsf{then}\ s\ \mathsf{else}\ s \end{array}$$

For expressions, we have numbers, booleans, variable references, arithmetic, an equality operator, and an if expression. There is also a do expression that runs a statement and then produces the result of the following expression.

For statements, we have assignment, sequential composition, and an if *statement*, which is distinct from the corresponding expression form.[8]

Here's an example program

```
if a = 3 then
  7
else  do if b = 2 then
             x := x + 1
           else
             y := y-1
      in
        x + x-y
```

The key point here is that the syntax of expressions depends on the syntax of statements, and vice-versa. This definitely happens in practice. The two are *defined mutually*.

[8] I believe that this distinction first appeared in Algol 58.

You may typically see such a set defined using rules like the following:

$$\frac{}{n \in \textsc{Expr}}\ (\text{num}) \qquad \ldots \qquad \frac{e_1 \in \textsc{Expr} \quad e_2 \in \textsc{Expr} \quad e_3 \in \textsc{Expr}}{\textsf{if } e_1 \textsf{ then } e_2 \textsf{ else } e_3 \in \textsc{Expr}}\ (\text{ife}) \qquad \frac{s \in \textsc{Stmt} \quad e \in \textsc{Expr}}{\textsf{do } s \textsf{ in } e \in \textsc{Expr}}\ (\text{do})$$

$$\ldots \qquad \frac{s_1 \in \textsc{Stmt} \quad s_2 \in \textsc{Stmt}}{s_1;s_2 \in \textsc{Stmt}}\ (\text{seq}) \qquad \ldots \qquad \frac{e \in \textsc{Expr}}{x \textsf{ := } e \in \textsc{Stmt}}\ (\text{asgn})$$

Some of these rules, like (ife) and (seq) look normal with matching syntactic sugar on top and bottom, but rules (asgn) and (do) look weird because they refer to different sets on top and bottom! What gives?!? Well, what really is happening is that we're defining a single set we'll call SE that essentially contains both EXP and STMT, in a way that we can tell in each case which set the other came from.

Let's desugar this definition. To tell the two sets apart, we introduce two atoms expr and stmt to tag elements of the sets. Now with these we can present the desugared inductive rules for SE.

Here is the totally desugared and fixed version of these rules.

$$\frac{}{\langle n, \textsf{expr}\rangle}\ (\text{num}) \qquad \ldots \qquad \frac{\langle r_1, \textsf{expr}\rangle \quad \langle r_2, \textsf{expr}\rangle \quad \langle r_3, \textsf{expr}\rangle}{\langle \textsf{if } r_1 \textsf{ then } r_2 \textsf{ else } r_3, \textsf{expr}\rangle}\ (\text{ife}) \qquad \frac{\langle r_1, \textsf{stmt}\rangle \quad \langle r_2, \textsf{expr}\rangle}{\langle \textsf{ do } r_1 \textsf{ in } r_2, \textsf{expr}\rangle}\ (\text{do}) \qquad \ldots$$

$$\frac{\langle r_1, \textsf{stmt}\rangle \quad \langle r_2, \textsf{stmt}\rangle}{\langle r_1;r_2, \textsf{stmt}\rangle}\ (\text{seq}) \qquad \ldots \qquad \frac{\langle r, \textsf{expr}\rangle}{\langle x \textsf{ := } r, \textsf{stmt}\rangle}\ (\text{asgn})$$

Notice that we got rid of references to e and s since they don't exist yet. Instead, we use tags to mark set elements.

Then, once we have the set SE, we can tease out our two desired sets.

$$\textsc{Expr} = \{\, r \in \textsc{Tree} \mid \langle r, \textsf{expr}\rangle \in \textsc{SE} \,\}$$
$$\textsc{Stmt} = \{\, r \in \textsc{Tree} \mid \langle r, \textsf{stmt}\rangle \in \textsc{SE} \,\}$$

Now that we have an inductive definition of SE, the inductive and recursive principles that we get for free are for SE, not for STMT, and not for EXPR alone.

However, in most cases we can act as though they are. Suppose a property P_s of statements s, and a property P_e of expressions. Then we can build a property

$$P(se) \equiv (se = \langle r, \textsf{expr}\rangle \wedge P_e(r)) \vee (se = \langle r, \textsf{stmt}\rangle \wedge P_s(r)).$$

If we plug this property into the Principle of Induction for elements of SE, then we get:

Proposition 9 (Principle of Mutual Induction for Elements $e \in \textsc{Expr}$ and $s \in \textsc{Stmt}$)**.** *Let P_s be a property of statements s, and P_e be a property of expressions. Then $P_s(s)$ holds for all $s \in \textsc{Stmt}$ and P_e holds for all $e \in \textsc{Expr}$ if:*

- *$P_e(n)$ for all $n \in \mathbb{Z}$;*
- *If $P_e(e_i)$ for $i \in \{1,2,3\}$ then $P_e(\textit{if } e_1 \textit{ then } e_2 \textit{ else } e_3)$;*
- *...*
- *If $P_s(s)$ and $P_e(e)$ then $P_e(\textit{ do } s \textit{ in } e)$;*
- *If $P_s(s_i)$ for $i \in \{1,2\}$ then $P_s(s_1;s_2)$;*
- *...*
- *If $P_e(e)$ then $P_s(x \textsf{ := } e)$.*

When one proves a property of a language by mutual induction, one is appealing to a proof of the above form.

Similarly, one can produce a Principle of Mutual Recursion, in its most general form, by following a similar pattern.

Proposition 10 (Principle of Definition by Mutual Recursion for $e \in$ EXPR and $s \in$ STMT)**.** *Suppose some set Q_s and some set Q_e and a collection of functions:*

- $E_n : n \to Q_e$;
- $E_{if} : Q_e \times Q_e \times Q_e \to Q_e$;
- ...
- $E_{do} : Q_s \times Q_e \to Q_e$;
- $S_; : Q_s \times Q_s \to Q_s$;
- ...
- $S_{:=} :$ VAR $\times Q_e \to Q_s$

Then there are unique functions $F_s :$ STMT $\to Q_s$ and $F_e :$ EXPR $\to Q_e$ such that:

- $F_e(n) = E_n(n)$;
- $F_e(\textit{if}\ e_1\ \textit{then}\ e_2\ \textit{else}\ e_3) = E_{if}(F_e(e_1), F_e(e_2), F_e(e_3))$;
- ...
- $F_e(\ \textit{do}\ s\ \textit{then}\ e) = E_{do}(F_s(s), F_e(e))$;
- $F_s(s_1;s_2) = S_;(F_s(s_1), F_s(s_2))$;
- ...
- $F_s(x := e) = S_{:=}(x, F_e(e))$.

The trick is to set $Q = (Q_s \times \{\,\textsf{stmt}\,\}) \cup (Q_e \times \{\,\textsf{expr}\,\})$ and build a function $F :$ SE $\to Q$ using the given parts. Then you can tease out of this the two functions F_s and F_e and prove that they exist and then prove that they are unique by mutual induction.

As a concrete example, consider the following definition of a pair of size functions for this language:

$$
\begin{aligned}
&\mathit{size}_e :: \text{EXPR} \to \mathbb{N}\\
&\mathit{size}_s :: \text{STMT} \to \mathbb{N}\\
&\mathit{size}_e(n) = 1\\
&\mathit{size}_e(\textsf{if}\ e_1\ \textsf{then}\ e_2\ \textsf{else}\ e_3) = \mathit{size}_e(e_1) + \mathit{size}_e(e_2) + \mathit{size}_e(e_3)\\
&\qquad\qquad \ldots\\
&\mathit{size}_e(\ \textsf{do}\ s\ \textsf{then}\ e) = \mathit{size}_s(s) + \mathit{size}_e(e)\\
&\mathit{size}_s(s_1;s_2) = \mathit{size}_s(s_1) + \mathit{size}_s(s_2)\\
&\qquad\qquad \ldots\\
&\mathit{size}_s(x := e) = \mathit{size}_e(e) + 1
\end{aligned}
$$

This counts as a definition of the two functions thanks to the principle of definition by mutual recursion for STMT and EXPR.