Chapter 21

Choose Your Own Induction Principle

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.

21.1 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

$$b \in B$$
$$b \quad ::= \quad \mathsf{t} \mid \mathsf{f} \mid b \bullet b$$

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:

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

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:

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

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 $(\bullet)$:

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

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

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

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:

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

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.

21.1.1 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 $\textsc{Tree}$s, 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 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

[1]ideally without confusing you...

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.

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

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 $\mathcal{D} \in \text{DERIV}[\mathcal{R}_2]$ such that $\mathcal{D} :: b$, so $B \subseteq C$. And furthermore, the rules $\mathcal{R}_2$ ensure that $C \subseteq B$. Thus,

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

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 79. *Let Φ be a property of derivations $D :: b$, then*

$$\begin{aligned}
&\Phi\left(\frac{}{\mathsf{t}}\right)\\
\wedge\ &\Phi\left(\frac{}{\mathsf{f}}\right)\\
\wedge\ &\left(\forall b_1, b_2 \in B.\, \forall D_1, D_2 \in \text{DERIV}[\mathcal{R}_2].\, \mathcal{D}_1 :: b_1 \wedge \mathcal{D}_2 :: b_2 \wedge \Phi(\mathcal{D}_1) \wedge \Phi(\mathcal{D}_2) \Rightarrow \Phi\left(\frac{\begin{matrix}\mathcal{D}_1 & \mathcal{D}_2\\ b_1 & b_2\end{matrix}}{b_1 \bullet b_2}\right)\right)\\
\Rightarrow &\forall D \in \text{DERIV}[\mathcal{R}_2].\Phi(D).
\end{aligned}$$

Notice how there are now *no* references to $r \in \text{TREE}$? 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.

21.2 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 80 (Principle of Mathematical Induction)**.** *Let $\Phi(n)$ be a property of natural numbers. Then Φ holds for all natural numbers if:*

1. *$\Phi(0)$ holds, and*

2. *If $\Phi(n)$ holds then $\Phi(n+1)$ holds.*

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

Proposition 81 (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 35 (1.). Then the sum of numbers from 0 to 0 is 0, which is the same as $\frac{0^2+0}{2}$.

Case 36 (2.). Let n be some particular number, and suppose $\Phi(n)$, 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:

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

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 82 (Principle of Strong Mathematical Induction)**.** *Let $\Phi(n)$ be a property of natural numbers. Then Φ holds for all natural numbers if:*

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

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 83 (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 37 $(n < m)$. Let $q = 0$ and $r = n$. Then $n = qm + r$ and we know by assumption that $r < m$, so this case is done.

Case 38 $(n \geq m)$. 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, $n - m = q_2m + r_2$ for some q_2 and $r_2 < m$. Then by adding m to both sides, we get:

$$\begin{aligned} n &= q_2m + r_2 + m \\ &= (q_2 + 1)m + r_2. \end{aligned}$$

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:

$$\forall n, m \in \mathbb{N}.\, m > 0 \Rightarrow \exists q, r \in \mathbb{N}.\, n = qm + r \wedge r < m. \tag{21.1}$$

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:

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

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]

21.3 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:

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

where $\mathbf{<_1 n}$ 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 84 (Principle of Mathematical Induction)**.** *Let $\Phi(n)$ be a property of natural numbers. Then Φ holds for all natural numbers if:*

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

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 39 $(n = 0)$*.* Then the sum of numbers from 0 to 0 is 0, which is the same as $\frac{0^2+0}{2}$.

Case 40 ($n = m + 1$ for some $m \in \mathbb{N}$)*.* Then by our assumption (i.e. the induction hypothesis), $\Phi(i)$, that is ...

$\square$

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)!

21.3.1 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 11 (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 85 (Principle of Well-founded Induction)**.** *Let S be a set, let $\sqsubset$ be a well-founded relation on S, and let $\Phi(s)$ be a relation on S. Then $\Phi(s)$ holds for all $s \in S$ if*

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

Formally:

$$(\forall s \in S.\,(\forall s' \in S.s' \sqsubset s \Rightarrow \Phi(s')) \Rightarrow \Phi(s)) \Rightarrow \forall s \in S.\,\Phi(s) \tag{21.2}$$

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

21.3.2 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 12. *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: $n <_{\text{fib}} m \quad \text{iff} \quad m = n+1 \vee m = n+2$. 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 13. *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 $\mathit{size} : \textsc{Term} \to \mathbb{N}$ that classifies the size of a term. Given such a function, it's easy to define a well-ordering on terms:

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

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 86 (Principle of Induction on $\mathit{size}(t)$)**.** *Let $\Phi(t)$ be a property of* TERMS. *Then $\Phi(t)$ holds for all terms t if the following is true: If $\Phi(t')$ holds for all terms t' such that $\mathit{size}(t') < \mathit{size}(t)$ then $\Phi(t)$ 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.

21.4 Well-founded Recursion: Recursive Definitions Unchained!

So far, we've described how "re-defining" a set can be used to explain many of the induction principles that appear in the literature, and empower you to develop your own induction principles, which may simplify your proof problems. But throughout the course we have also used induction principles as a means to justify definitions of recursive functions (namely functions whose definite descriptions consist of propositions that fundamentally involve equations wherein reference to function itself sometimes appears on both sides of the

[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.

equations). You might wonder if redefinition can empower us with new recursion principles, and the answer is *yes*!

In particular, we are going to describe "the mother of all (induction-induced) recursion principles": *well-founded recursion.* I think of this as the mother of all recursion principles in the sense that it captures in an abstract and quite general way all of the recursion principles we have seen so far. In particular, we can quite directly prove each of the previous induction-induced recursion principles we have seen using this one: just a little book-keeping is required.

This principle is phrased in terms of some non-descript well-founded relation, which we can instantiate to match many of our previous inductive definitions, but with some manual labour to determine how to split things into cases, as we did with our proofs by well-founded induction.

Let's state the principle, and then spend some time unpacking it. Then we can see an example of shoehorning a previous recursion principle into the structure of well-founded recursion so you can see the pattern, and recognize the flexibility. Bear in mind: well-founded recursion is quite the bazooka, so in practice you should only use it when its flexibility is necessary: structural or primitive recursion will more clearly convey many many function definitions. Constraints can be clarifying.

Proposition 87 (Principle of Well-founded Recursion)**.**

1. *Let I be a set, $(\bullet \sqsubset \bullet) \subseteq I \times I$ be a well-founded relation, and* $\sqsubset\mathbf{i} := \{\, i_o \in I \mid i_0 \sqsubseteq i \,\}$;
2. *Let S be some set;*
3. *Let $H_I : D \to S$, where* $D := \{\, \langle i, f\rangle \; \boldsymbol{for}\, i \in I, f \in \sqsubset\mathbf{i} \to S \,\}$;

Then there exists a unique function $F : I \to S$ such that

$$F(i) = H_I\left(i, F|_{\sqsubset\mathbf{i}}\right)$$

where $F|_{\sqsubset\mathbf{i}}$ is the restriction *of F to the domain $\sqsubset\mathbf{i}$, i.e. the unique function $G : \sqsubset\mathbf{i} \to S$ such that $G(i) = F(i)$.*

Well, that's a mouthful! It's also not the easiest proposition to make heads or tails of given all the layers of abstraction. Let's take it in parts.

First, step 1 essentially picks an inductively defined set. Or, more specifically, it takes some set I, defined however the heck you want, and a well-founded relation that empowers us to *re-define* I inductively. So this corresponds to the usual "consider some inductively-defined set", albeit without actually saying it.

Step 2 is where we commit to the codomain S of the function that we are setting out to define. So at lest this part should be comfortable.

Step 3 is where things start to get crazy. Instead of identifying particular elements s of S, one for each "base case", and enumerating a collection of functions H, one for each rule schema from an inductive definition, we identify *one big honking crazy function* H_I! And this function has a pretty wild domain D. But at least, like our old principles, the codomain of this function is S.

Let's examine this domain D in detail. It is a set of element-function pairs. In particular each element i is paired with a function from all elements that are "less than" i (remember, $\sqsubset$ need not be an ordering relation, just well-founded, hence the scare quotes). So for any particular i, the set D will have one entry for every possible function that you can form from $\sqsubset\mathbf{i}$ to S. That can be a lot of functions, specifically $|S|^{|\sqsubset\mathbf{i}|}$ of them! To summarize, D has $|S|^{|\sqsubset\mathbf{i}|}$ pairs for each $i \in I$. Good thing :: slaps on JZF :: this bad boy can hold so many.

Here's where I recommend that you think of this domain as a big ole spreadsheet of spreadsheets. Each entry $\langle i, f\rangle$ has in its first column an element i of the domain, and the second column is a reference to another sheet that associates one element of S to each element of $\sqsubset\mathbf{i}$: one for each smaller element.

Now let's consider the function H_I. This function has one job. Given one row of the spreadsheet, i.e. given one $\langle i, f\rangle$ pair, it needs to associate that to an element s of S. *In principle* such a function can be maximally flexible, taking into account every last detail of i itself, *and* every last detail of each individual $\langle i_0, s\rangle$ entry in the accompanying referenced spreadsheet f. So H_I has *lots and lots* of wiggle room. Furthermore, you (the forger of new functions $F : I \to S$ have access to *the full power of set theory* to define an H_I. You don't always need such great power, and to be honest, the vast majority of the time you just need a teensy bit of it beyond the simpler recursion principles. So its expressiveness scales with your needs.

Now that we've talked about the inputs to the principle, it's time to discuss what great payoff we have reaped! The output of the principle is a proof that there exists a unique function $F : I \to S$ that is determined by a single equation involving H_I. What does this equation tell us? I'll describe it two different ways.

First, let's go with the mathematical perspective, which is purely proposition-driven. One almost taken-for-granted facet of this statement is the principle that given *any* function $G : A \to B$ and *any* subset $C \subseteq A$, there is a unique function $G\big|_C : C \to B$, the restriction of G to the domain C. So in practice we're talking about a whole lot of functions, but we can view this as the outcome of selecting a bunch of rows of a spreadsheet and copying them to a new sheet to create a new "smaller" function from an old one. So for any Function $F : I \to S$ we can associate any $i \in I$ to a function $F|_{\sqsubset \mathbf{i}}$. Bear in mind that two elements $i_o \neq i_1$ may induce the same set $\sqsubset\mathbf{i_0} = \sqsubset\mathbf{i_1}$. The most obvious case is minimal elements for which $\sqsubset\mathbf{i} = \emptyset$, which are all associated with $F|_\emptyset$[8]

Ok, so given any function $F : I \to S$, we get a unique set of restricted functions $F|_{\sqsubset \mathbf{i}}$. The final part of the theorem, and the part that's a little mind-blowing is that given any $H_I : D \to S$, there exists a function that satisfies $F(i) = H_I\left(i, F|_{\sqsubset \mathbf{i}}\right)$, and moreover, any two such functions are identical! In short, H_I uniquely determines (a.k.a. "describes") F. The proof of this looks a lot like the proof of the principle of structural recursion, but gets a little more jazzy because it uses well-founded induction (naturally, since this is a principle of well-founded recursion) and the assumption that H_I exists.

So that was the "declarative" mathematical definition. Now for an "algorithmic" explanation, for which all of the usual caveats apply. Given a particular element i, we can imagine being able to determine the collection $\sqsubset\mathbf{i}$. Furthermore, we can imagine that a "call" to $H_I(i, f)$ will use its first argument i to decide which elements of $f(i_0)$ are needed, based on $i_0 \sqsubset i$, to produce the appropriate result s. Those elements can be thought of as "recursive function calls", so long as you dont have to worry that in principle there may be an infinite number of them[9]

Ultimately the real work in using the principle of well-founded recursion amounts to defining the right H_I to get you what you want. Don't let the full power of H_I confuse you into thinking that defining it is just as hard as defining F. This is analogous to the difference between defining a set of rule instances and using them to inductively define a set: the latter is the "complex" goal that the former facilitates.

21.4.1 Well-founded Sheep in Structural Clothing

To get some first insight into well-founded recursion, let's consider the principle of *primitive* recursion for the B set from the first section and we will show how to use the principle of well-founded recursion to prove it. This will show some direct connections. It should also be clear how the principle of structural recursion relates too, since it just has fewer components than the principle of primitive recursion.

Proposition 88 (Principle of Primitive Recursion for $b \in B$)**.** *Let:*

1. *S be a set;*
2. *s_t be an element of S;*
3. *s_f be an element of S;*
4. *$H_\bullet : B \times B \times S \times S \to S$ be a total function.*

Then there is a unique function $F : B \to S$ such that

1. *$F(\mathsf{t}) = s_t$;*
2. *$F(\mathsf{f}) = s_f$;*
3. *$F(b_1 \bullet b_2) = H_\bullet(b_1, b_2, F(b_1), F(b_2))$.*

[8]Yes, for every set B there is a unique function $\emptyset \to B$: it's $\emptyset$! Think about it.

[9]That's right, $\sqsubset\mathbf{i}$ can be infinite. This corresponds to having an infinitely "wide" derivation tree that still has finite "height".

Our strategy for sketching a proof of the above proposition is to define an appropriate well-founded relation, and then apply the principle of well-founded recursion. Our well-founded relation $(\bullet \sqsubset_B \bullet) \subseteq B \times B$ is defined by the two rules:

$$\frac{}{b_1 \sqsubset_B b_1 \bullet b_2} \qquad\qquad \frac{}{b_2 \sqsubset_B b_1 \bullet b_2}$$

Since we already have some set S, then we can determine the domain D for the $H_B : D \to S$ function.

$$\begin{aligned} D &= \{\, \langle b, f\rangle \textbf{ for } b \in B, f \in \sqsubset\mathbf{b} \to S \,\} \\ &= \{\, \langle \mathsf{t}, \emptyset\rangle \,\} \cup \{\, \langle \mathsf{f}, \emptyset\rangle \,\} \cup \{\, \langle b_1 \bullet b_2, f\rangle \textbf{ for } b_1 \in B, b_2 \in B, f \in \{\, b_1, b_2 \,\} \to S \,\}\,. \end{aligned}$$

We can view the domain as having three distinct kinds of pairs:

1. A pair of t with the unique total function $f : \emptyset \to S$, namely $\emptyset$. If you check the requirements for being a total function from some set A to some set B, you will find that $\emptyset$ is always a legitimate total function if the domain is $\emptyset$. This takes getting used to, but remember: an empty spreadsheet is a perfectly fine spreadsheet!

2. A pair f with $\emptyset$. This is similar to the above. An interesting thing about the empty total function is that, unlike most total functions you've seen *you can't even call this one*. That doesn't mean it's *not* a function, it just means that it contains no interesting information, except that it has entries for all zero elements of B that are below f.

3. A pair of $b_1 \bullet b_2$ with a function $f : \{\, b_1, b_2 \,\} \to S$. But remember that a function is just a spreadsheet, so we could rewrite f as $f = \{\, \langle b_1, s_1\rangle, \langle b_2, s_2\rangle \,\}$.

Now that we've seen the structure of D, we can define a function H_B that produces a unique function, and then double-check that the resulting function satisfies the equations from Prop. 88 (and only one function can do that).

$$\begin{aligned} H_B(\mathsf{t}, \emptyset) &= s_t \\ H_B(\mathsf{f}, \emptyset) &= s_f \\ H_B(b_1 \bullet b_2, \{\, \langle b_1, s_1\rangle, \langle b_2, s_2\rangle \,\}) &= H_\bullet(b_1, b_2, s_1, s_2) \end{aligned}$$

There are a couple of interesting things happening up above. First, everything is quite uniform: even the cases for t and f are paired with a function, albeit a vacuous one. Second, we can see quite clearly that the "function" second argument is really just a tabulation of component elements and their corresponding elements, but as a set of pairings rather than a quadruple that lists all of the component elements before the corresponding elements. The only difference is the structure of the encoding. The main difference is that the function-based encoding is much more flexible, and can handle an arbitrary number of component elements. The use of domain D allows us to describe how the number and nature of premise components depends explicitly on the first element of the pair.

If we give H_B to the principle of well-founded recursion, we get a unique function $F : B \to S$ that satisfies $F(b) = H_B\left(b, F|_{\sqsubset\mathbf{b}}\right)$, which when analyzed by cases on B gives us:

$$\begin{aligned} F(\mathsf{t}) &= H_B\left(\mathsf{t}, F|_{\sqsubset\mathbf{tb}}\right) = H_B(\mathsf{t}, \emptyset) = s_t \\ F(\mathsf{f}) &= H_B\left(\mathsf{f}, F|_{\sqsubset\mathbf{fb}}\right) = H_B(\mathsf{f}, \emptyset) = s_t \\ F(b_1 \bullet b_2) &= H_B\left(b_1 \bullet b_2, F|_{\sqsubset(\mathbf{b_1 \bullet b_2})}\right) = H_B(b_1 \bullet b_2, \{\, \langle b_1, s_1\rangle, \langle b_2, s_2\rangle \,\}) = H_\bullet(b_1, b_2, s_1, s_2) \end{aligned}$$

Voila: we've recovered the primitive recursion equations! Technically we are still on the hook to prove that at most one function can satisfy the primitive recursion equations, since all we've done is prove that the well-founded recursive function also satisfies these, but not that these are sufficient to determine the function. We are not going to worry about that here though.

One interesting difference between H_B and $H_\bullet$ is that $H_\bullet$ is not applied to the entire expression $b_1 \bullet b_2$, but H_B is. That's because $H_\bullet$ could reconstruct the input $b_1 \bullet b_2$ from b_1 and b_2, so in this case what we have is simply an optimization. However this kind of reconstruction is *not possible* in general when defining a function by well-founded recursion. A well-founded relation may relate arbitrarily different elements, so long as all descending chains are finite. In particular, two different elements may have the same set of "smaller" elements. So the general solution is to pass along the element itself as an argument. It plays the role that we typically play by supplying distinct subscripted elements and function inputs, but now everything is wrapped up into H_B.

21.4.2 A Nontrivial Use of Well-founded Recursion

We just showed that the principle of well-founded recursion can be used to define functions that we already know how to define, using structural or primitive recursion principles. But we did that mostly to help us understand how the principle works: what we provide and how it gives us a proper function definition.

Now let's use well-founded recursion to justify a function that might otherwise be challenging to define. For this one we choose a classic, due to the logician Wilhelm Ackermann.[10] Speaking anachronistically, Ackermann observed that the following function on natural numbers is total in the sense we use in this class (that for any input we can deduce an output), but its description does not fit into the template laid forth by primitive recursion. His function was as follows:[11]

$$\begin{aligned}
\mathcal{A} &: \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to N \\
\mathcal{A}(m, n, 0) &= m + n \\
\mathcal{A}(m, 0, 1) &= 0 \\
\mathcal{A}(m, 0, 2) &= 1 \\
\mathcal{A}(m, 0, p+3) &= m \\
\mathcal{A}(m, n+1, p+1) &= \mathcal{A}(m, \mathcal{A}(m, n, p+1), p)
\end{aligned}$$

That this function is defined over a product $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$ suffices for it to not *directly* be defined by primitive induction over numbers. However, primitive recursion with respect to a particular coordinate-wise ordering scheme, where only one coordinate determines order, is often treated as though it were synonymous to primitive recursion with respect to $\mathbb{N}$. Morally speaking, the extra elements of the product are "parameters" that remain constant and do not affect the propriety argument for the definite description.

However, single-coordinate primitive recursion does not suffice to justify this function, since the last equation depends on the value of $\mathcal{A}$ for a decrease in the third argument (for the outer self-referential function application) as well as a decrease in the second argument (for the inner self-reference). To make matters worse, the outer self-reference depends on the value of the inner self-reference, which may denote *any* natural number whatsoever! So even a more general notion of coordinate-wise recursion does not suffice.

To justify this function definition, the following well-founded relation on $\mathbb{N}^3$ suffices:

$$\frac{}{\langle m, n, p \rangle \sqsubset_{\mathcal{A}} \langle m, n+1, p \rangle} \qquad \frac{}{\langle m, o, p \rangle \sqsubset_{\mathcal{A}} \langle m, n, p+1 \rangle}$$

This relation embeds a 2-dimensional lexicographic ordering into $\mathbb{N}^3$, where the third position dominates the second: if p decreases, then n may change arbitrarily. We could have more precisely tailored the order to suit the $\mathcal{A}$ function specifically, but it is often useful to reuse existing orderings (or their embedding among other arguments), because then well-foundedness is already assured.

[10]See `https://en.wikipedia.org/wiki/Ackermann_function`

[11]The binary function that we today call "the Ackermann function" was actually adapted from this function by Rósza Péter (the "founding mother of recursion theory") and Raphael Robinson (husband of the great Julia Robinson).

Now we can justify A using the function:

$$\begin{aligned}
H_{\mathsf{A}} &: D \to \mathbb{N} \\
H_{\mathsf{A}}(\langle m, n, 0\rangle, f) &= m + n \\
H_{\mathsf{A}}(\langle m, 0, 1\rangle, f) &= 0 \\
H_{\mathsf{A}}(\langle m, 0, 2\rangle, f) &= 1 \\
H_{\mathsf{A}}(\langle m, 0, p+3\rangle, f) &= m \\
H_{\mathsf{A}}(\langle m, n+1, p+1\rangle, f) &= f(m, f(m, n, p+1), p)
\end{aligned}$$

We have omitted the definition of the domain D since it is schematic. The function H_{A} looks *a lot* like the A function, but *it is not recursive*. Each case makes use of the function f that was passed in.

Technically, we are on the hook to justify that H_{A} is a total function $D \to \mathbb{N}$, but this is not too difficult. We can prove that the four equations cover all possible elements of D, since each applies no additional constraint to the function f (don't forget that each of these functions has a different domain, determined by the number triple that appears alongside it!). Proving this is often called *coverage checking* or *totality* checking. Second, we can prove that the three equations are *non-overlapping*, in the sense that each applies for a distinct set of possible triples from N^3, and describes a total function over its own subset. This property is not *strictly* necessary, but it greatly simplifies the proof that H_{A} is a function in that one need not prove that overlapping equations are mutually consistent for the arguments over which they overlap. One power of mathematical functions is that it's perfectly fine to have an overlapping equation like $H_{\mathsf{A}}(\langle m, n, p\rangle, f) = H_{\mathsf{A}}(\langle m, n, p\rangle, f)$, which overlaps with *all of the equations* so long as the entire collection still yields a proper definite description of a total function. But such equations make manual reasoning (and automated reasoning by way of implementing a decision *procedure*) more difficult.

Part of our reasoning about each equation invoves ensuring that any use of the function argument f applies it to elements of the function's domain. In particular, in the last equation $f \in A \to \mathbb{N}$ where:

$$A = \{\, \langle m', n', p'\rangle \in \mathbb{N}^3 \mid \langle m', n', p'\rangle \sqsubset_{\mathcal{A}} \langle m, n+1, p+1\rangle \,\}\,.$$

We can prove that $\langle m, n, p+1\rangle \sqsubset_{\mathcal{A}} \langle m, n+1, p+1\rangle$ immediately using the first rule. Then, since $\langle m, o, p\rangle \sqsubset_{\mathcal{A}} \langle m, n, p+1\rangle$ for any $o \in \mathbb{N}$ whatsoever, and $f(\langle m, n, p+1\rangle) \in \mathbb{N}$, we can use the second rule to prove that $\langle m, f(m, n, p+1), p\rangle \sqsubset_{\mathcal{A}} \langle m, n+1, p+1\rangle$.

From here, we can apply equational reasoning to the results of the Principle of Well-founded Recursion to prove that we have indeed constructed the $\mathcal{A}$ function.

21.5 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.

21.5.1 Preconditions vs. Side Conditions

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

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

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 $\longrightarrow^*$:

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

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 $t_1 \rightarrow t_2$, 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:

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

Is this definition of $\skull \subseteq \textsc{Term} \times \textsc{Term}$ alright? Sure! the natural definition of $\not\longrightarrow$ is as follows:

$$\not\longrightarrow \; = \{\, p \in \textsc{Term} \times \textsc{Term} \mid p \notin \longrightarrow \,\}$$

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:

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

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

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.

21.5.2 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.

$$\begin{array}{lcl}
 & & n \in \mathbb{Z}, \quad x \in \textsc{Var}, \quad e \in \textsc{Expr}, \quad s \in \textsc{Stmt} \\
e & ::= & n \mid \mathsf{true} \mid \mathsf{false} \mid x \mid e \mathbin{\mathsf{+}} e \mid e \mathsf{-} e \mid e \mathbin{\mathsf{=}} e \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid \ \mathsf{do}\ s\ \mathsf{in}\ e \\
s & ::= & x \mathrel{::==} e \mid s \mathsf{;} s \mid \mathsf{if}\ e\ \mathsf{then}\ s\ \mathsf{else}\ s
\end{array}$$

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.[12]

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

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*.

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

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

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

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.

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

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

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.

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

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 Φ_s of statements s, and a property Φ_e of expressions. Then we can build a property

$$\Phi(se) \equiv (se = \langle r, \mathsf{expr}\rangle \wedge \Phi_e(r)) \vee (se = \langle r, \mathsf{stmt}\rangle \wedge \Phi_s(r)).$$

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

Proposition 89 (Principle of Mutual Induction for Elements $e \in \textsc{Expr}$ and $s \in \textsc{Stmt}$)**.** *Let Φ_s be a property of statements s, and Φ_e be a property of expressions. Then $\Phi_s(s)$ holds for all $s \in \textsc{Stmt}$ and Φ_e holds for all $e \in \textsc{Expr}$ if:*

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

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 90 (Principle of Definition by Mutual Recursion for $e \in \textsc{Expr}$ and $s \in \textsc{Stmt}$)**.** *Suppose some set Q_s and some set Q_e and a collection of functions:*

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

Then there are unique functions $F_s : \textsc{Stmt} \to Q_s$ and $F_e : \textsc{Expr} \to Q_e$ such that:

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

The trick is to set $Q = (Q_s \times \{\,\mathsf{stmt}\,\}) \cup (Q_e \times \{\,\mathsf{expr}\,\})$ and build a function $F : \mathrm{SE} \to Q$ 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:

$$\begin{aligned}
&\mathit{size}_e :: \textsc{Expr} \to \mathbb{N}\\
&\mathit{size}_s :: \textsc{Stmt} \to \mathbb{N}\\
\mathit{size}_e(n) &= 1\\
\mathit{size}_e(\mathsf{if}\; e_1 \;\mathsf{then}\; e_2 \;\mathsf{else}\; e_3) &= \mathit{size}_e(e_1) + \mathit{size}_e(e_2) + \mathit{size}_e(e_3)\\
&\dots\\
\mathit{size}_e(\,\mathsf{do}\; s \;\mathsf{then}\; e) &= \mathit{size}_s(s) + \mathit{size}_e(e)\\
\mathit{size}_s(s_1;; s_2) &= \mathit{size}_s(s_1) + \mathit{size}_s(s_2)\\
&\dots\\
\mathit{size}_s(x ::== e) &= \mathit{size}_e(e) + 1
\end{aligned}$$

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

[RG: Some stuff from Spring 2013]

In a recent homework assignment, I gave a definition of a mathematical function

$$\mathit{append} : \text{ECtxt} \times \text{ECtxt} \to \text{ECtxt}$$

for combining two valuation contexts. I defined it as follows:

$$\begin{aligned}
append(E, \square) &= E \\
append(E_1, E_2[\square + t]) &= append(E_1, E_2)[\square + t] \\
append(E_1, E_2[v + \square]) &= append(E_1, E_2)[v + \square] \\
append(E_1, E_2[\textsf{if}\ \square\ \textsf{then}\ t_1\ \textsf{else}\ t_2]) &= append(E_1, E_2)[\textsf{if}\ \square\ \textsf{then}\ t_1\ \textsf{else}\ t_2]
\end{aligned}$$

Recall that earlier in the term I said that a set of propositions (like the above equations) only counts as a definition if we establish that they indeed state the properties of a single object. In this case, how do we know that there is a single function $f \in \text{ECTXT} \times \text{ECTXT} \to \text{ECTXT}$ that satisfies these equations? You might be tempted to say "by the principle of definition by recursion on the structure of contexts E_2", and you'd be pretty much right, but not completely. In particular, the principle evoked above is for defining functions $f \in \text{ECTXT} \to S$ for some set S, but *append* above has as its domain the set of *pairs* of evaluation contexts, not single contexts. So what gives?

21.6 Induction on Terms, Generalized

Throughout the course we have been using inductive rules to define sets and prove properties of them. Here we take a step back to the early days to note how one of our earlier ideas generalizes.

First some review. In Lecture 3, we used inductive rules to define the syntax of the Boolean language, building on the set of TREES, as below.

$$\text{(r-true)}\frac{}{\textsf{true} \in \text{TERM}} \qquad \text{(r-false)}\frac{}{\textsf{false} \in \text{TERM}} \qquad \text{(r-if)}\frac{r_1 \in \text{TERM} \quad r_2 \in \text{TERM} \quad r_3 \in \text{TERM}}{\textsf{if}\ r_1\ \textsf{then}\ r_2\ \textsf{else}\ r_3 \in \text{TERM}}$$

Based on the structure of the definition, we knew that $\text{TERM} \subseteq \text{TREE}$, since the definition was essentially carving out a subset of trees explicitly.

We next introduced a principle of induction on the structure of derivation, followed by a principle of induction on the structure of TERMS. Here I repeat that principle, but with a slight modification from the earlier statement.

Proposition 91 (Principle of Structural Induction on terms $t \in \text{TERM}$).
Let $\Phi(r)$ be a property of TREEs r. Then Φ holds for all TERMs t if:

1. *$\Phi(\textsf{true})$ holds;*
2. *$\Phi(\textsf{false})$ holds;*
3. *If $\Phi(t_1)$ $\Phi(t_2)$, and $\Phi(t_3)$ hold then $\Phi(\textsf{if}\ r_1\ \textsf{then}\ r_2\ \textsf{else}\ r_3)$ holds.*

The statement is a generalization of our earlier statement. The only difference here is that Φ is a property of TREES, not TERMS. Of course that means that Φ is also a property of TERMS, but it's well defined for all TREES. This change means that we can apply the principle of induction for more general properties, without worrying about whether the element under question is specifically a TERM. Below you'll see why we care about this.

21.6.1 Two Inductions for the Multi-Step Relation

In a recent homework, we gave rules to define two relations, $\longrightarrow^* \subseteq \text{TERM} \times \text{TERM}$ and $\longrightarrow^{\square} \subseteq \text{TERM} \times \text{TERM}$.

Here are the rules that define $\longrightarrow^*$:

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

and here are the rules that define $\longrightarrow^{\square}$:

$$\text{(zero)}\frac{}{t \longrightarrow^{\square} t} \qquad \text{(plus-1)}\frac{t_2 \longrightarrow^{\square} t_3}{t_1 \longrightarrow^{\square} t_3}\ t_1 \longrightarrow t_2$$

You were asked to state the corresponding principles of induction for each of these relations, and to prove that $\longrightarrow^*$ and $\longrightarrow^\square$ are in fact one and the same relation! So we end up with two different definitions and names for the same set.

With this realization may come a little concern: does it matter which mathematical definition we use for this set? In particular, if we use the $\longrightarrow^\square$ definition, are we banned from using the induction principle that we get from $\longrightarrow^*$? The answer is a resounding *NO*! To make this point clearer, let's state the principles of induction that are analogous to the principle of induction on the structure of terms:

Proposition 92 (Principle of Rule Induction ($t \longrightarrow^* t$))**.**
Let $\Phi(t,t)$ be a property of pairs of terms. Then $\Phi(t_1,t_2)$ holds for all $t_1 \longrightarrow^* t_2$ if:

1. *If $t_1 \longrightarrow t_2$ then $\Phi(t_1,t_2)$ holds;*
2. *$\Phi(t,t)$ holds; and*
3. *If $\Phi(t_1,t_2)$ holds and $\Phi(t_2,t_3)$ holds, then $\Phi(t_1,t_3)$ holds.*

Proposition 93 (Principle of Rule Induction ($t \longrightarrow^\square t$))**.**
Let $\Phi(t,t)$ be a property of pairs of terms. Then $\Phi(t_1,t_2)$ holds for all $t_1 \longrightarrow^* t_2$ if:

1. *$\Phi(t,t)$ holds; and*
2. *If $\Phi(t_2,t_3)$ holds and $t_1 \longrightarrow t_2$ then $\Phi(t_1,t_3)$ holds.*

Here, something interesting happens. When we stated the definition of induction on the structure of terms, it looks plainly like we are talking about how a particular term is built: maybe start with a true, and a false and another true and build up aif true then false else true term from it. But now we're just talking about pairs of terms, t_1, t_2. They aren't "built" up based on either of these principles, but each of these principle is guided by the inductive rules that we used to define the set. For this reason, we call the above principles instances of *rule induction*, because the structure of the terms and the rules used to define them don't coincide (but they did when we defined the terms, so the distinction was lost).

Notice that both Principles are stated in terms of $\longrightarrow^*$ now: $\longrightarrow^\square$ is only mentioned in the name of one principle so that you can see where the principle came from. Now, we can use either of our definitions, whichever one seems to capture our intuitive notion of multi-step, but we get to use keep both of these principles. In this case, I choose $\longrightarrow^*$ as our definition because it very explicitly captures the idea of "reflexive-transitive closure of $\longrightarrow$". There's one rule for reflexivity and one rule for transitivity, and one rule for what's being closed over. If my formal definition matches my intuitive definition, then I'm more likely to have captured what I meant to. Only then do we need to worry about what reasoning principles we want.[13]

21.7 Defining *append* manually

In the last section, we learned how to get new induction principles for an old set: give a new definition for it. In fact, this is essentially how we can justify the definition of the *append* function. Usually we take the set ECtxt × ECtxt for granted, since we can always form pairs, but we can also provide a new definition for pairs:

$$\frac{}{E \times []} \qquad \frac{E_1 \times E_2}{E_1 \times E_2[[] + +t]} \qquad \frac{E_1 \times E_2}{E_1 \times E_2[v + +[]]} \qquad \frac{E_1 \times E_2}{E_1 \times E_2[\textsf{if}\ []\ \textsf{then}\ t_2\ \textsf{else}\ t_3]}$$

The above rules define the same set ECtxt × ECtxt, but they give us a new induction principle for the same set. What's further, the rule set is *deterministic*, meaning that there is exactly one derivation tree for every member of the set. This is particularly easy to tell in this case, because for each member of the set, only one rule could possibly at the bottom of the derivation. Whenever this is true, you immediately get a Principle of Definition by Recursion.[14]

[13]As an example of this broader point: note that you learn about Turing Machines when you are taught about the notion of computation, because a Turing machine intuitively feels like a good formalization of computation. That they are equivalent to lambda calculus gives you another way of reasoning though, hence the Church-Turing Thesis.

[14]When your rules are not deterministic, like for $\longrightarrow^*$, you need extra constraints to ensure *coherence* of your definition, meaning that it's really a function, with one result for each input.

Proposition 94. *Let*

1. S *be some set;*
2. $H_{[]} : \textsc{ECtxt} \to S$*;*
3. $H_{[]++t} : S \times \textsc{Term} \to S$*;*
4. $H_{v++[]} : S \times \textsc{Value} \to S$*;*
5. $H_{\mathit{if}} : S \times \textsc{Term} \times \textsc{Term} \to S$*.*

Then there is unique function $F : \textsc{ECtxt} \times \textsc{ECtxt} \to S$ *such that:*

1. $F(E, \Box) = H_{\Box}(E)$*;*
2. $F(E_1, E_2[\Box + t]) = H_{\Box + t}(F(E_1, E_2), t)$
3. $F(E_1, E_2[v + \Box]) = H_{v + \Box}(F(E_1, E_2), v))$
4. $F(E_1, E_2[\textit{if}\,\Box\ \textit{then}\ t_1\ \textit{else}\ t_2]) = H_{\mathit{if}}(F(E_1, E_2), t_1, t_2)$.

This is principle of recursion that we use to define *append*. I'll leave it as an exercise for the reader to deduce the definitions of S and the various H functions.

You have to admit, it's a bit of a pain that we would have to actually come up with a new definition every time we want a principle of induction or a principle of recursion, and have to ensure that the definition is deterministic too. Is there an easier way?

Bibliography

P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, chapter C.7, pages 739–782. North-Holland, 1977.

J. Avigad. Reliability of mathematical inference. *Synthese*, 2020. doi: 10.1007/s11229-019-02524-y. `https://doi.org/10.1007/s11229-019-02524-y`.

F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, NY, USA, 1998. ISBN 0-521-45520-0.

J. Bagaria. Set theory. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* The Metaphysics Research Lab, winter 2014 edition, 2014. URL `http://plato.stanford.edu/archives/win2014/entries/set-theory/`.

R. Baldoni, E. Coppa, D. C. D'elia, C. Demetrescu, and I. Finocchi. A survey of symbolic execution techniques. *ACM Comput. Surv.*, 51(3):50:1–50:39, May 2018. ISSN 0360-0300. doi: 10.1145/3182657. URL `http://doi.acm.org/10.1145/3182657`.

H. P. Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics.* North-Holland, 1985. ISBN 978-0-444-86748-3.

A. Bauer. Proof of negation and proof by contradiction. Mathematics and Computation Blog, March 2010. `http://math.andrej.com/2010/03/29/proof-of-negation-and-proof-by-contradiction/`.

L. Crosilla. Set Theory: Constructive and Intuitionistic ZF. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* Metaphysics Research Lab, Stanford University, Summer 2020 edition, 2020.

M. Felleisen and D. P. Friedman. A syntactic theory of sequential state. *Theoretical Computer Science*, 69(3):243–287, 1989. ISSN 0304-3975. doi: https://doi.org/10.1016/0304-3975(89)90069-8. URL `http://www.sciencedirect.com/science/article/pii/0304397589900698`.

M. Felleisen, R. B. Findler, and M. Flatt. *Semantics Engineering with PLT Redex.* MIT Press, 1st edition, 2009.

J. Ferreirós. The early development of set theory. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* Metaphysics Research Lab, Stanford University, summer 2019 edition, 2019. URL `https://plato.stanford.edu/archives/sum2019/entries/settheory-early/`.

D. Grossman, G. Morrisett, T. Jim, M. Hicks, Y. Wang, and J. Cheney. Region-based memory management in cyclone. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation*, PLDI '02, pages 282–293, New York, NY, USA, 2002. ACM. ISBN 1-58113-463-0. doi: 10.1145/512529.512563. URL `http://doi.acm.org/10.1145/512529.512563`.

P. R. Halmos. *Naive Set Theory.* Springer-Verlag, first edition, Jan. 1960. ISBN 0387900926.
A classic introductory textbook on set theory.

P. R. Harper. *Practical Foundations for Programming Languages.* Cambridge University Press, New York, NY, USA, 2012. URL `http://www.cs.cmu.edu/%7Erwh/plbook/1sted-revised.pdf`.

D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124 (2):103–112, Feb. 1996. ISSN 0890-5401.

G. Kahn. Natural semantics. In *4th Annual Symposium on Theoretical Aspects of Computer Sciences on STACS 87*, pages 22–39, London, UK, UK, 1987. Springer-Verlag.

R. Krebbers. *The C standard formalized in Coq*. PhD thesis, Radboud University Nijmegen, December 2015. URL `https://robbertkrebbers.nl/thesis.html`.

C. Kuratowski. Sur la notion de l'ordre dans la théorie des ensembles. *Fundamenta Mathematicae*, 2(1): 161–171, 1921. doi: 10.4064/fm-2-1-161-171. `https://web.archive.org/web/20190429103938/http://matwbn.icm.edu.pl/ksiazki/fm/fm2/fm2122.pdf`.

P. J. Landin. The mechanical evaluation of expressions. *Comput. J.*, 6(4):308–320, 1964. doi: 10.1093/comjnl/6.4.308. URL `https://doi.org/10.1093/comjnl/6.4.308`.

X. Leroy and H. Grall. Coinductive big-step operational semantics. *Inf. Comput.*, 207(2):284–304, Feb. 2009. ISSN 0890-5401.

P. Maddy. Believing the axioms. I. *The Journal of Symbolic Logic*, 53(02):481–511, 1988.
An interesting (though complicated) analysis of why set theorists believe in their axioms.

J. McCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *Commun. ACM*, 3(4):184–195, 1960.

J. H. J. Morris. *Lambda-Calculus Models of Programming Languages*. PhD thesis, Massachusetts Institute of Technology, Feb. 1969. URL `http://hdl.handle.net/1721.1/64850`.

A. M. Pitts. *Nominal sets: Names and symmetry in computer science*. Cambridge University Press, 2013.

G. D. Plotkin. The origins of structural operational semantics. *J. Log. Algebr. Program.*, 60-61:3–15, 2004a. doi: 10.1016/j.jlap.2004.03.009. URL `http://dx.doi.org/10.1016/j.jlap.2004.03.009`.

G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004b.

B. Popik. "pull yourself up by your bootstraps". Weblog entry, September 2012. `https://www.barrypopik.com/index.php/new_york_city/entry/pull_yourself_up_by_your_bootstraps/`.

E. Reck and G. Schiemer. Structuralism in the Philosophy of Mathematics. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2020 edition, 2020.

D. Sangiorgi. On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4): 15:1–15:41, May 2009. ISSN 0164-0925.

W. Sieg and D. Schlimm. Dedekind's analysis of number: Systems and axioms. *Synthese*, 147(1):121–170, Oct 2005.

J. Spolsky. The law of leaky abstractions. Joel on Software Blog, November 2002. `https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/`.

G. L. Steele. Debunking the "expensive procedure call"" myth or, procedure call implementations considered harmful or, lamdba: The ultimate goto. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1977. URL `http://dspace.mit.edu/handle/1721.1/5753`.

S. Stenlund. Descriptions in intuitionistic logic. In S. Kanger, editor, *Proceedings of the Third Scandinavian Logic Symposium*, volume 82 of *Studies in Logic and the Foundations of Mathematics*, pages 197 – 212. Elsevier, 1975. URL `http://www.sciencedirect.com/science/article/pii/S0049237X08707328`.

M. Tiles. Book Review: Stephen Pollard. Philosophical Introduction to Set Theory. *Notre Dame Journal of Formal Logic*, 32(1):161–166, 1990.
A brief introduction to the philosophical issues underlying set theory as a foundation for mathematics.

C. Urban and M. Norrish. A formal treatment of the Barendregt variable convention in rule inductions. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Mechanized Reasoning about Languages with Variable Binding*, MERLIN '05, page 25–32, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930728. doi: 10.1145/1088454.1088458. URL `https://doi.org/10.1145/1088454.1088458`.

C. Urban, S. Berghofer, and M. Norrish. Barendregt's variable convention in rule inductions. In *Proceedings of the 21st International Conference on Automated Deduction: Automated Deduction*, CADE-21, page 35–50, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 9783540735946. doi: 10.1007/978-3-540-73595-3_4. URL `https://doi.org/10.1007/978-3-540-73595-3_4`.

D. van Dalen. *Logic and structure (3. ed.)*. Universitext. Springer, 1994. ISBN 978-3-540-57839-0.

A. K. Wright and M. Felleisen. A syntactic approach to type soundness. *Inf. Comput.*, 115(1):38–94, Nov. 1994.

B. Zimmer. figurative "bootstraps". email to linguistlist mailing list, August 2005. `http://listserv.linguistlist.org/pipermail/ads-l/2005-August/052756.html`.