Reasoning about Inductive Definitions: Inversion, Induction, and Recursion

CPSC 509: Programming Language Principles

Ronald Garcia*

13 January 2014
(**Time Stamp**: 23:22, Sunday 14th October, 2018)

Previously, we defined the small Vapid programming language. Since the language has a finite number of programs, its syntax was very easy to define: just list all the programs! In turn it was straightforward to define its evaluation function by cases, literally enumerating the results for each individual program. Finally, since the evaluator was defined by listing out the individual cases (program-result pairs), we could prove some (not particularly interesting) properties of the language and its programs.[1]

In an effort to move toward a more realistic language, we have introduced the syntax of a language of Boolean expressions, which was more complex than Vapid in that there are an infinite number of Boolean expressions. We did this using inductive definitions, which are much more expressive and sophisticated than just listing out programs. However, we must now answer the question: how do we define an evaluator for this infinite-program language, and how can we prove properties of *all* programs in the language and the results of evaluating them? To answer this question, we introduce three new reasoning principles: inversion lemmas, proofs by induction, and definitions of functions by recursion.

1 Derivation and Inversion

Recall the definition of the language of Boolean Expressions $t \in \text{TERM} \subseteq \text{TREE}$:

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

From the above, we know that $\text{TERM} = \{ r \in \text{TREE} \mid \exists \mathcal{D}.\, \mathcal{D} :: r \in \text{TERM} \}$. Remember that the $\in \text{TERM}$ on the right side of the comprehension is syntactic sugar, so this is not circular. I sometimes leave off that particular piece of sugar because it looks problematic in this context. So we know that for each element $t \in \text{TERM}$, it is also true that $t \in \text{TREE}$; furthermore, there must be *at least* one derivation $\mathcal{D} :: r \in \text{TERM}$. More formally, our definition of the set TERM gives us the reasoning principle:

$$\forall t. t \in \text{TERM} \iff t \in \text{TREE} \wedge \exists \mathcal{D} \in \text{DERIV}. D :: t \in \text{TERM}.$$

We take advantage of this connection between derivations and TERMs to reason about the TERMs by proving things about derivations. First, we take the following property of our derivations as given.

Proposition 1 (Principle of Cases on Derivations $\mathcal{D} :: r \in \text{TERM}$).
For all $\mathcal{D} \in \text{DERIV}$, $r \in \text{TREE}$, if $\mathcal{D} :: r \in \text{TERM}$, then exactly one of the following is true:

1. $\mathcal{D} = \dfrac{}{\mathsf{true} \in \text{TERM}}$ *(r-true)*

[1] In general, these language properties are interesting, but because Vapid is so...vapid, the properties are trivial.

2. $\mathcal{D} = \dfrac{}{\textit{false} \in \textsc{Term}}$ *(r-false)*

3. $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1\\ r_1 \in \textsc{Term}\end{array} \quad \begin{array}{c}\mathcal{D}_2\\ r_2 \in \textsc{Term}\end{array} \quad \begin{array}{c}\mathcal{D}_3\\ r_3 \in \textsc{Term}\end{array}}{\textit{if}\ r_1\ \textit{then}\ r_2\ \textit{else}\ r_3 \in \textsc{Term}}$ *(r-if)* *for some derivations* $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$.

Based on our understanding of inductive rules and derivations that use instances of these rules, the above statement is obviously true. We state it explicitly for two reasons.

First, though we choose to take this proposition as given directly from our inductive definition, this is not strictly true. I've mentioned that we can build the idea of inductive rules and derivations directly in set theory, where an inductive rule is some kind of set, and a definition is another kind of set, etc. If we were to do this, then the above principle would be a *theorem* of set theory, not an *fact* that we take as given. However, diving that deep is too much like programming directly in machine language: the bits that your CPU understands. At the least we can use something more akin to assembly language (a smidge higher-level than machine language) as our starting point, which makes life a little easier, albeit not quite as easy as we like. For this reason we will build new easier principles on top of this, but we'll know how to hand-compile our statements down to the "assembly language" level. This can be helpful (at least it has been for me) when it comes to understanding whether what you have written down actually makes mathematical sense.

Second, which is related to the first, we need to start somewhere. We need some "rules of the game" to work with. Taking this principle as given is a nice starting point in my opinion. If you'd like to see what the bottom looks like, I can point you toward some further reading. Instead, just assume that whenever you have an inductive definition, you get a principle of cases on the structure of derivations.

1.1 Inversion Lemmas

So now that we have this principle of cases, what can we do with it? Well, we can prove a set of lemmas about TERMs, that you will expect to be obviously true, but might not have been sure how to prove.

Lemma 1 (Inversion on $r \in \textsc{Term}$)**.** *If* $r \in \textsc{Term}$ *then one of the following is true:*

1. $r = \textit{true}$

2. $r = \textit{false}$

3. $r = \textit{if}\ r_1\ \textit{then}\ r_2\ \textit{else}\ r_3$ *for some* $r_1, r_2, r_3 \in \textsc{Term}$.

Proof. Since this is one of the first proofs you'll see, I'm going to walk through it in painful detail, then rewrite it as you would typically see in a paper. The first is to help you understand the strategy, and the second is to help your writing.

Notice that we are proving an implication: **if** $r \in \textsc{Term}$, **then** case 1 holds or case 2 holds or case 3 holds. To prove an implication, we assume the premise and try to prove the conclusion, so:

"Suppose some $r \in \textsc{Term}$."

Now we need to prove the consequence: "case 1 holds or case 2 holds or case 3 holds."

To make progress, we observe that $r \in \textsc{Term}$ can only be true if there is some $\mathcal{D} :: r \in \textsc{Term}$ (this is the reasoning princple that we get from the axiom schema of separation). So we have deduced the existence of *some* $\mathcal{D}$ that fits the bill. Here we *could* say "Then there is a derivation $\mathcal{D}$", but typically we don't bother in writing out these proofs. Most paper proofs skip over this observation because "by goodness WE KNOW that you're using inductive definitions so WE KNOW there's a derivation: get on with it!" But it's useful to recognize, even if you don't write it down, that you are taking a logical step here.

Next, now that we have a derivation $\mathcal{D}$, (since we've concluded that one exists!) we apply the Principle of Cases on Derivations to our derivation to conclude that one of the following is true.

1. $\mathcal{D} = \dfrac{}{\text{true} \in \textsc{Term}}$ (r-true)

2. $\mathcal{D} = \dfrac{}{\text{false} \in \textsc{Term}}$ (r-false)

3. $\mathcal{D} = \dfrac{\dfrac{\mathcal{D}_1}{r_1 \in \textsc{Term}} \quad \dfrac{\mathcal{D}_2}{r_2 \in \textsc{Term}} \quad \dfrac{\mathcal{D}_3}{r_3 \in \textsc{Term}}}{\mathsf{if}\ r_1\ \mathsf{then}\ r_2\ \mathsf{else}\ r_3 \in \textsc{Term}}$ (r-if) for some derivations $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$.

So we now know that one of the above 3 things is true, and we want to show that one of the three original things up above holds: we are using one disjunction to prove another. So as with our small model of propositional logic, we use (or *eliminate*) a disjunction by separately assuming each of the three cases and trying to prove the conclusion. On the other end, we can establish (or *introduce*) a disjunction by proving any one of the disjuncts. We don't need to prove all of them, otherwise we'd actually be proving a conjunction. So the usual prose for using a disjunction is to say something like:

"We proceed by cases on the structure of $\mathcal{D}$"

And then write out the cases separately as follows.

Case. Suppose $\mathcal{D} = \dfrac{}{\mathsf{true} \in \textsc{Term}}$ (r-true). Then clearly $\mathcal{D} :: \mathsf{true} \in \textsc{Term}$, so $r = \mathsf{true}$. Thus we have proven that one of the three conclusions holds.

Case. Suppose $\mathcal{D} = \dfrac{}{\mathsf{false} \in \textsc{Term}}$ (r-false) Then clearly $\mathcal{D} :: \mathsf{false} \in \textsc{Term}$, so $r = \mathsf{false}$. Thus we have proven that one of the three conclusions holds.

Case. Suppose $\mathcal{D} = \dfrac{\dfrac{\mathcal{D}_1}{r_1 \in \textsc{Term}} \quad \dfrac{\mathcal{D}_2}{r_2 \in \textsc{Term}} \quad \dfrac{\mathcal{D}_3}{r_3 \in \textsc{Term}}}{\mathsf{if}\ r_1\ \mathsf{then}\ r_2\ \mathsf{else}\ r_3 \in \textsc{Term}}$ (r-if) for some derivations $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$. Well clearly $r = \mathsf{if}\ r_1\ \mathsf{then}\ r_2\ \mathsf{else}\ r_3 \in \textsc{Term}$, and from the three subderivations we deduce that $r_1, r_2, r_3 \in \textsc{Term}$.

□

Notice that all the way down, the structure of the proof was analogous to the structure of proofs in our small formal model of propositional logic. What I haven't formally presented is how to introduce or eliminate $\exists$ or $\forall$ in CPL. Ideally we can avoid formalizing those, but rather get more comfortable with them through practice.

First, let me rewrite this proof as I would normally write it down. This mostly involves leaving out details that a seasoned theorem-prover will be able to fill in herself.

Proof. Suppose $r \in \textsc{Term}$. We then proceed by cases on the structure of $\mathcal{D}$

Case ($\mathcal{D} = \dfrac{}{\mathsf{true} \in \textsc{Term}}$ (r-true)). Then $r = \mathsf{true}$ immediately.

Case ($\mathcal{D} = \dfrac{}{\mathsf{false} \in \textsc{Term}}$ (r-false)). Analogous to the previous case.

Case ($\mathcal{D} = \dfrac{\dfrac{\mathcal{D}_1}{r_1 \in \textsc{Term}} \quad \dfrac{\mathcal{D}_2}{r_2 \in \textsc{Term}} \quad \dfrac{\mathcal{D}_3}{r_3 \in \textsc{Term}}}{\mathsf{if}\ r_1\ \mathsf{then}\ r_2\ \mathsf{else}\ r_3 \in \textsc{Term}}$ (r-if)). Then $r = \mathsf{if}\ r_1\ \mathsf{then}\ r_2\ \mathsf{else}\ r_3$ immediately, and from the three subderivations we deduce that $r_1, r_2, r_3 \in \textsc{Term}$.

□

Now for some explanation. Roughly speaking, an inversion lemma is just a way of saying that if the conclusion of an inductive rule holds, then the premises of the rule hold as well. In general, things get more complex, especially because an inductive definition may have two different derivations for the same element of the defined set (e.g. from the entailment relation, $\{\top\} \vdash \top$ **true**: I leave it to you to find two derivations). Nonetheless, there is a corresponding notion of inversion lemmas in this case, but it may merge rules that can produce the same result. Usually we won't bother proving these inversion lemmas, because the proof is always done the same way, and you expect it to be true. Nonetheless, these lemmas are used *a lot* going forward so you should know how to prove them.

Second, these inversion lemmas are sometimes useful when thinking about implementing artifacts related to the proof rules. In the case of $r \in \textsc{Term}$, what we basically get is a strategy for *implementing a parser* for Terms. Essentially, for our purposes, a parser is a program that given some tree r, tries to build a derivation $\mathcal{D} :: r \in \textsc{Term}$ starting from the bottom and working upwards. If you look at the lemmas, we can see that at each point, the next step of searching is relatively clear. When we introduce more sophisticated inductive definitions like relations for evaluating programs or for specifying which programs are well-typed, we will specialize the inversion lemmas to assume certain inputs (e.g. input program) and yield certain outputs (e.g., the result of evaluation).

2 Inductive Reasoning

The strategy that we use to define an evaluator and prove properties about it follows our ongoing theme that *the structure of your definitions guides the structure of your reasoning*. In the case at hand, we defined the syntax of the Boolean Expressions using an *inductive definition*, which consisted of a set of *inductive rules*, whose *instances* could be used to build *derivations* that "prove" which TREEs we want to accept as members of the set of TERMs. For our purposes, an inductive definition "automatically" gives us reasoning principles tailored to the particular definition, just like each axiom of set theory gives us a reasoning principle that we can work with.The reasoning principle for our inductively defined set TERM follows.

Proposition 2 (Principle of Structural Induction on Derivations $\mathcal{D} :: r \in \text{TERM}$)**.**
Let P be a predicate on derivations $\mathcal{D}$. Then $P(\mathcal{D})$ holds for all derivations $\mathcal{D}$ if:

1. $P\left(\dfrac{}{\mathsf{true} \in \text{TERM}}\ \textit{(r-true)}\right)$ *holds;*
2. $P\left(\dfrac{}{\mathsf{false} \in \text{TERM}}\ \textit{(r-false)}\right)$ *holds;*
3. *For all $\mathcal{D}_1, D_2, D_3 \in \text{DERIV}$, $r_1, r_2, r_3 \in \text{TREE}$, such that $\mathcal{D}_i :: r_i$,*
   If $P\begin{pmatrix} \mathcal{D}_1 \\ r_1 \in \text{TERM} \end{pmatrix}$, $P\begin{pmatrix} \mathcal{D}_2 \\ r_2 \in \text{TERM} \end{pmatrix}$, *and* $P\begin{pmatrix} \mathcal{D}_3 \\ r_3 \in \text{TERM} \end{pmatrix}$ *hold then*

$$P\left(\dfrac{\begin{array}{ccc} \mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_3 \\ r_1 \in \text{TERM} & r_2 \in \text{TERM} & r_3 \in \text{TERM} \end{array}}{\mathsf{if}\ r_1\ \mathsf{then}\ r_2\ \mathsf{else}\ r_3 \in \text{TERM}}\ \textit{(r-if)}\right)$$

   holds.

Proof. For our purposes, this comes for free with the inductive definition of $t \in \text{TERM}$. □

Whenever we define a set using inductive rules, we get a principle of induction on the derivations from those rules. These principles all have the same general structure: assume that you have some property P of derivations. Then that property holds for all derivations if for *each* rule in the inductive rule, the property holds for a derivation of the conclusion if it held for the derivations of each of the premises. This can be most clearly seen in case 3. above, where the property holds for the 3 subderivations of the if derivation and then holds for the whole derivation itself. The first two cases are a little different. Since the (r-true) and (r-false) rules have no premises, it's *vacuously* true that the property holds for all of the subderivations: because there are none.

Without delving into an actual proof of this, the intuition is this: In a sense, this theorem is a recipe for building up a proof that $P(\mathcal{D})$ holds for any particular $\mathcal{D}$: If we know that P holds for any derivation that is exactly an axiom, and we know that whenever we combine derivations $\mathcal{D}_i$ that satisfy P using some rule, we get a single tree that also satisfies P, then we can take *any* derivation, tear it apart, and prove that the leafs of the tree (at the top) satisfy P and we can systematically put the tree back together, proving at each step that the resulting piece satisfies P, until we've finally rebuilt the entire original tree and established that indeed $P(\mathcal{D})$ holds.

2.1 Aside: Reasoning about Function Definitions

So how do we use this principle of induction in practice? Below we will apply it to deduce facts about a function, but we'll first spend some time using tools we already had to reason about a function. Then we'll see an example of using induction to prove something that we already *know* to be true, but that we didn't have the tools to formally *prove*. Note that this is the typical progression in math: as an intuitive human, we have the sense that something is true, but then we prove it formally so as to be sure. Sometimes we're surprised to discover that the thing we "knew" is false. So much for being perfect!

Anyway let's start with a simple function. Consider the following function definition:

Definition 1. *Let* $bools : \text{TERM} \to \mathbb{N}$ *be defined by*

$$
\begin{aligned}
bools(\textit{true}) &= 1 \\
bools(\textit{false}) &= 1 \\
bools(\textit{if } t_1 \textit{ then } t_2 \textit{ else } t_3) &= bools(t_1) + bools(t_2) + bools(t_3)
\end{aligned}
$$

Remember that an equational function definition like the above ought be interpreted roughly as follows:

$$
\text{Let } S = \left\{ F \in \text{TERM} \to \mathbb{N} \;\middle|\; \begin{array}{c} F(\text{true}) = 1 \wedge \\ F(\text{false}) = 1 \wedge \\ \forall t_1, t_2, t_3 \in \text{TERM}. F(\text{if } t_1 \text{ then } t_2 \text{ else } t_3) = F(t_1) + F(t_2) + F(t_3) \end{array} \right\}.
$$
$$
\text{then } bools \in S \text{ and } \forall f \in \text{TERM} \to \mathbb{N}. f \in S \Rightarrow f = bools.
$$

We can restate it as a proposition, which makes it clearer that there is technically an obligation to prove that proposition:

$$
\exists! bools \in \text{TERM} \to \mathbb{N}. \quad \begin{array}{c} bools(\text{true}) = 1 \wedge \\ bools(\text{false}) = 1 \wedge \\ \forall t_1, t_2, t_3 \in \text{TERM}. bools(\text{if } t_1 \text{ then } t_2 \text{ else } t_3) = bools(t_1) + bools(t_2) + bools(t_3) \end{array}
$$

In short, there is a *unique* function in $\text{TERM} \to \mathbb{N}$ that satisfies the three propositions, and we will use the name $bools$ for it. Technically we are on the hook to prove that (1) there exists *at least one* function that satisfies those three equations; and (2) there exists *at most one* function that satisfies those three equations. For now, let's assume that both of these claims are true. Below, we will prove that a particular *template* for equations is guaranteed to define a unique function, and that the equations for $bools$ fit that template. We call that template the Principle of Recursion on $t \in \text{TERM}$.

Anyway, we have a function now and we'll assume it exists. What do we know about it *right now*? Well we know by the axiom of separation that (1) it assigns a unique natural number to each term t, and (2) that these assignments are constrained by the equations given above. In fact, since this is a function *definition*, we know that these constraints are sufficient to determine the function entirely.

First, let's prove something we already know about how $bools$ treats one particular term.

Proposition 3. $bools(true) = 1$.

Proof. $bools$ satisfies three equations, and conveniently the first one immediately completes the proof. □

Not a very satisfying proof, but a proof nonetheless! Remember: $bools$ is technically just a particular subset of $\text{TERM} \times \mathbb{N}$, in fact an infinite subset, but nonetheless we can use the few facts that we know about it to deduce facts. In short, we use *equational reasoning* to deduce new facts about $bools$. It's a bit absurd for me to write the above as a proposition and a proof: no one does that in real life, they just write out the calculation as in the following example:

$$
\begin{aligned}
& bools(\text{if false then true else true}) \\
=& bools(\text{false}) + bools(\text{true}) + bools(\text{true}) \\
=& 1 + 1 + 1 \\
=& 3.
\end{aligned}
$$

Each step of the above appeals to equational reasoning to learn something. In the above example, we used our sparse knowledge from the definition to deduce *additional* facts about the $bools$ function. You may be rolling your eyes a bit, since you have been doing stuff like this since secondary school, but it's useful to recognize this as an instance of *deduction*: learning new facts from old. All too often we think of a function as a machine that performs calculations given inputs. Hogwash! Our function is just a table of mappings from TERM to $\mathbb{N}$. It sits there like a dead fish. Not only that, it's infinite, so we can't hope to just read down the list of entries to find the one that we want (I'm pretty sure it wouldn't fit on disk anyway, let alone in memory). Instead it is *we*, the logical deduction engines, that use *only* the fact that the function exists, as well as a few other properties of the function, in particular the equational constraints, to *deduce* some of the entries in this infinite table. In short, functions don't compute, they are just facts: *calculation* is the process of deducing facts about a function. That's what we use computers for.

Okay, enough with the philosophy for now. Let's get back to deducing facts about $bools$. In the above cases, we used equational reasoning to deduce facts about particular entries. Now, let's deduce facts about entire *classes* of entries.

Proposition 4. $\forall t \in \text{TERM}.bools(\textit{if}\ t\ \textit{then}\ t\ \textit{else}\ t) = 3 * bools(t)$.

Proof.

$$\begin{aligned}&bools(\text{if } t \text{ then } t \text{ else } t)\\ =&bools(t) + bools(t) + bools(t)\\ =&3 * bools(t).\end{aligned}$$

□

This proposition is *not* that much different looking from the last one, but it's way more general: the last one told us a fact about *one* term, while this one tells us something about *an infinite number of terms*! So exciting. In practice, if I were on the hook to implement $bools$ as a computer program (i.e. a deduction engine for this function), I could possibly exploit this fact to add an optimization to the deduction engine: "if you see a case like this, don't bother computing $bools(t)$ three times: just do it once and multiply." As programmers, we make these steps of deduction all the time as we work. In essence, this is what an optimizing compiler (or interpreter!) for a programming language does too. Here we make such a deduction explicit and justify it with a formal proof of its correctness.

2.2 Your First Proof By Induction

In the last section, we showed that in the set-theoretic world, an equational function definition is just a "constraint filter" on the set of functions with a domain. Note that I am specifically saying *equational* function definition, because there are other ways to define functions (i.e., pick out an individual element of the set of functions). And we showed that we can use those equations to deduce new properties of the function...these deductions are exactly the calculations that we perform to determine the value of a function for a particular input. But we also show that those same deductions can be used determine facts about *entire* classes of values, and given such a deduction we can optimize future deductions. We'll find out later that people write computer programs that automatically perform these kinds of deductions too. For instance, the subfield of program analysis called *symbolic execution* Baldoni et al. [2018] is precisely about writing an automatic and efficient "proof engine" for deducing useful facts about abstract expressions, where *symbols* play the role of our metavariables. The workaday programmer is much more comfortable with writing such deduction engines for the value of a function when given a concrete input! Shockingly enough, we call such deduction engines...wait for it..."functions."

Hopefully these side-commentaries about deduction might help you make a useful observation: many computer programs that you and others are writing can be viewed as *automated theorem provers* for very limited classes of theorems. In the case of implementing $bools$ as a programming language function (which I sometimes call a *procedure* to disambiguate), the corresponding program accepts a term t, uses deduction like above to prove the specialized theorem $bools(t) = n$, *throws away the proof*, and just gives you the number n.

Time for a theorem! Now it's worth noticing something interesting about our $bools$ function if only intuitively: The function is defined to evaluate to *natural numbers* $\mathbb{N}$, but not every natural number has some term to which $bools$ maps it. In particular, there is no term such that $bools(t) = 0$! How do we know? Well, just by staring at the equations and saying something like "look, the number cases use 0 and the if case uses +..." But how do we *formally* prove that this hand-wavy explanation is correct? Clearly we can *test* this hypothesis by coming up with a bunch of concrete TERMs and deducing their values and then gaining confidence in our observation. But since we have an infinite number of TERMs, we cannot exhaustively test the $bools$ function. Another way to say this last sentence is: we cannot appeal to the reasoning principle that we get by giving an extensional definition of a finite set like $\{ a, b, c \}$ because TERM is infinite, so we *can't* give an extensional definition.

So what to do? Use the Principle of Induction on derivations $\mathcal{D} :: r \in \text{TERM}$! And you'll find yourself using induction principles time and time again in this course.

Let's work through this proof, in more painful of detail than you would see in the literature. It's more important to understand what is really going on first, then see how people take shortcuts when writing it down.

Proposition 5. *$bools(t) > 0$ for all $t \in \text{TERM}$.*

Proof. A note about the proof statement. Mathematicians sometimes put the quantifiers (for all, for some) at the *end* of the statement, even though it appears at the beginning of a formal presentation: $\forall t \in \text{TERM}.bools(t) > 0$ Other times they leave off the quantifiers, then you have to guess what they mean. Both of these are meant to emphasize the most important part of the statement, since a person can usually figure out the proper quantification from context. Sadly, this doesn't always work: sometimes you are left scratching your head wondering exactly what they mean, and if the proof isn't there for you to inspect, you may end up sending an annoyed email to the author to find out what the heck they meant...sadly I've been that author some times. Don't be that author.

We are going to use the Principle of Induction, but that will not get us all the way to exactly the statement above, but it will get us most of the way there.

To use the principle, I first need to pick a particular *property* of derivations. Here it is:

$$P(\mathcal{D}) \equiv \forall t \in \text{TERM}.\mathcal{D} :: t \Rightarrow bools(t) > 0.$$

I'm justified *technically* to use $t \in \text{TERM}$ here because every TERM is a TREE, so the property is at least well-formed: it's always reasonable to ask if $\mathcal{D} :: t$ since $:: \subseteq \text{DERIV} \times \text{TREE}$. Furthermore, I'm justified *pragmatically* to restrict my property to TERMs t, in the sense that this will ultimately work out, because we already know by definition that if $t \in \text{TERM}$ then $\mathcal{D} :: t$ for *some* derivation $\mathcal{D}$, so if I can prove this property for all *derivations*, then it will imply the property I really care about for all *terms*. We'll see this arise as the last step of our proof.

Now I will specialize the conditions on the Principle of Induction according to this property, which yields 3 *lemmas*, minor propositions, that I will prove.

The first condition was $P\left(\dfrac{}{\mathsf{true} \in \text{TERM}}\ \text{(r-true)}\right)$, so plugging in my property yields the following condition:

Lemma 2.

$$\forall t \in \text{TERM}. \left(\frac{}{\mathsf{true} \in \text{TERM}}\ \textit{(r-true)}\right) :: t \Rightarrow bools(t) > 0.$$

Proof. Suppose $t \in Term$, and $\mathcal{D} :: t$. Then $t = \mathsf{true}$, and $bools(t) = bools(\mathsf{true}) = 1 > 0$. □

Okay, let's all acknowledge that I'm being rather pedantic in this proof. But the reason I'm doing so is just to show that there's no magic: we learned in the last subsection how to use equational reasoning to prove things, by plug-and-chug, and similarly here I just plugged my property into the definition verbatim and proved the theorem that it gave me. The annoying part is...why did I have to deal with this "for all t's that the derivation could possibly be a derivation of"? Well, because the general form of the property has to quantify over t, and only after we pick a particular derivation can we show that t must be exactly the same thing as the conclusion. Note that if we treated :: as a function from derivations to trees (which it is: $\mathcal{D} :: t$ would be the same as saying $::(\mathcal{D}) = t$), then I would still end up deducing that $\mathsf{true} = t$. In any case, all of that is detail, but ideally you see that we're being very systematic, much like a computer program would have to be. That rigour becomes helpful (and not merely tedious) only when the objects and proofs get more complicated, as well as when the prover or proof checker becomes a computer program.

Okay, I've now got two more lemmas to prove:

Lemma 3.

$$\forall t \in \textsc{Term}. \left(\frac{}{\textsf{false} \in \textsc{Term}} \; (\textit{r-false}) \right) :: t \Rightarrow \textit{bools}(t) > 0.$$

Proof. Suppose $t \in Term$, and $\mathcal{D} :: t$. Then $t = \textsf{false}$, and $\textit{bools}(t) = \textit{bools}(\textsf{false}) = 1 > 0$. □

Okay, let's be honest: I wrote the above proof by copying the previous one and replacing the trues with falses. That is to say, this proof follows essentially the same argument as the previous one. In a paper or tech report, you are more likely to see:

Proof. Analogous to the lemma for true. □

For communicating with experts this is a *good thing*: it tells me in a compressed form what the proof for this case is like, so I don't have to read through the details and discover that the proof is indeed analogous. However, it had better be true that the proof goes analogously! That is to say, better to have proven it, discover that it is analogous, and then compress the proof afterwards, rather than assume that it's analogous and be *wrong!* This is one of the pitfalls that people fall into when proving theorems. For at least the first part of the class, I will ask you to present each proof in full, so that you get more practice with proving each of the individual cases (even if "practice" means getting your cut-and-paste-and-edit right...).

Now for the last case, which is the most interesting one because it really demonstrates the power of induction:

Lemma 4. *For all $\mathcal{D}_1, D_2, D_3 \in \textsc{Deriv}$, $r_1, r_2, r_3 \in \textsc{Tree}$, such that $\mathcal{D}_i :: r_i$,*
If

$$\forall t \in \textsc{Term}. \left(\begin{array}{c} \mathcal{D}_1 \\ r_1 \in \textsc{Term} \end{array} \right) :: t \Rightarrow \textit{bools}(t) > 0,$$

and

$$\forall t \in \textsc{Term}. \left(\begin{array}{c} \mathcal{D}_2 \\ r_2 \in \textsc{Term} \end{array} \right) :: t \Rightarrow \textit{bools}(t) > 0,$$

and

$$\forall t \in \textsc{Term}. \left(\begin{array}{c} \mathcal{D}_3 \\ r_3 \in \textsc{Term} \end{array} \right) :: t \Rightarrow \textit{bools}(t) > 0$$

then

$$\forall t \in \textsc{Term}. \left(\frac{\begin{array}{c} \mathcal{D}_1 \\ r_1 \in \textsc{Term} \end{array} \quad \begin{array}{c} \mathcal{D}_2 \\ r_2 \in \textsc{Term} \end{array} \quad \begin{array}{c} \mathcal{D}_3 \\ r_3 \in \textsc{Term} \end{array}}{\textsf{if}\ r_1\ \textsf{then}\ r_2\ \textsf{else}\ r_3 \in \textsc{Term}} \; (\textit{r-if}) \right) :: t \Rightarrow \textit{bools}(t) > 0.$$

Wow, what a mouthful! But notice the structure of the argument: all we have to prove is that if the property holds for derivations $\mathcal{D}_i$ of the subtrees r_i , then we can build a proof that the property holds for the derivation that you get when you hook together the $\mathcal{D}_i$ derivations to form a proof $\mathcal{D}$ that if r_1 then r_2 else $r_3 \in \textsc{Term}$. Also, notice that the derivations $\mathcal{D}_i$ and r_i are quantified universally at the beginning. Then every reference to those names in the lemma refers to the same derivation. In contrast, each individual precondition has its own t that is quantified universally, so we can use these preconditions to deduce facts about a variety of t's. It turns out that for this proof we will use each precondition only once. Okay let's prove it!

Proof. Suppose $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3 \in \textsc{Deriv}$, and $r_1, r_2, r_3 \in Tree$, and $\mathcal{D}_i :: r_i \in \textsc{Term}$. Furthermore, suppose

$$\forall t \in \textsc{Term}. \left(\begin{array}{c} \mathcal{D}_i \\ r_i \in \textsc{Term} \end{array} \right) :: t \Rightarrow \textit{bools}(t) > 0$$

holds for each $\mathcal{D}_i :: r_i$.

Now let

$$\mathcal{D} = \left(\dfrac{\overset{\mathcal{D}_1}{r_1 \in \textsc{Term}} \quad \overset{\mathcal{D}_2}{r_2 \in \textsc{Term}} \quad \overset{\mathcal{D}_3}{r_3 \in \textsc{Term}}}{\textsf{if } r_1 \textsf{ then } r_2 \textsf{ else } r_3 \in \textsc{Term}} \text{ (r-if)} \right).$$

It suffices to show that

$$\forall t \in \textsc{Term}. \left(\overset{\mathcal{D}}{\textsf{if } r_1 \textsf{ then } r_2 \textsf{ else } r_3 \in \textsc{Term} \in \textsc{Term}} \right) :: t \Rightarrow \mathit{bools}(t) > 0,$$

so let's do it:

Suppose $t \in \textsc{Term}$ and $\mathcal{D} :: t$. Then $t = \textsf{if } r_1 \textsf{ then } r_2 \textsf{ else } r_3 \in \textsc{Term}$. Using our assumptions we can prove that $\mathit{bools}(r_i) > 0$.[2] Let me do it for one case. Apply the assumption

$$\forall t \in \textsc{Term}. \left(\overset{\mathcal{D}_1}{r_1 \in \textsc{Term}} \right) :: t \Rightarrow \mathit{bools}(t) > 0,$$

to the term r_1 to get:

$$\left(\overset{\mathcal{D}_1}{r_1 \in \textsc{Term}} \right) :: r_1 \Rightarrow \mathit{bools}(r_1) > 0.$$

By assumption, $\mathcal{D}_1 :: r_2$, which when applied to the above, gives us $\mathit{bools}(r_1) > 0$.[3] We can repeat this reasoning for r_2 and r_3.

From there, we can calculate

$$\mathit{bools}(\textsf{if } r_1 \textsf{ then } r_2 \textsf{ else } r_3) = \mathit{bools}(r_1) + \mathit{bools}(r_2) + \mathit{bools}(r_3).$$

But using our knowledge that $\mathit{bools}(r_i) > 0$, and our deeply-ingrained knowledge of how mathematics works, we see that $\mathit{bools}(r_1) + \mathit{bools}(r_2) + \mathit{bools}(r_3) > 0$. □

Cool, so so far, we've proven three interesting lemmas, two of which are about concrete derivations (r-true) and (r-false), and one that is about what happens if you build a derivation using (r-if) from three pre-existing derivations that satisfy our property P. Are we done?

Technically no! Now we *apply* the principle of induction on derivations to our three lemmas to get:

Lemma 5.

$$\forall \mathcal{D} \in \textsc{Deriv}. \forall t \in \textsc{Term}. \mathcal{D} :: t \Rightarrow \mathit{bools}(t) > 0.$$

Great, so now we know something about *all derivations* $\mathcal{D} \in \textsc{Deriv}$. Surely we're done? The pedant says no! Now, much like the induction hypotheses from earlier, we deduce from this fact *and* the definition of $\textsc{Term}$ the knowledge that we really want.

Proposition 6. $\forall t \in \textsc{Term}. \mathit{bools}(t) > 0$.

Proof. Suppose $t \in \textsc{Term}$. Then by the definition of $\textsc{Term}$, there exists some derivation $\mathcal{D} \in \textsc{Deriv}$ such that $\mathcal{D} :: t$. Applying the last lemma to that derivation, we get that $\forall t' \in \textsc{Term}. \mathcal{D} :: t' \Rightarrow \mathit{bools}(t') > 0$.[4] Well, I know that $t \in \textsc{Term}$, so if I apply the above proposition to it, I get that $(\mathcal{D}) :: t \Rightarrow \mathit{bools}(t) > 0$. And now, we already knew that $\mathcal{D} :: t$, so we apply the above proposition to this fact and we get, pant pant pant, *finally*, $\mathit{bools}(t) > 0$. Woohoo! □

□

[2] 2) This is where you often see a proof say "by the induction hypothesis"! The induction hypotheses are just the assumptions we made that the property applies to each subderivation-tree pair $\mathcal{D}_i, r_i$.

[3] Notice the terminology, that I'm "applying" a proposition to a "term"...sounds a lot like I'm applying a function to an argument, eh? This is no coincidence!

[4] Notice that without fanfare, I renamed the quantified t in the result to t' so that it doesn't cause us problems in a second. You can always rename quantified names, and sometimes it helps avoid confusion.

Okay, I walked through this proof in *painstaking detail* since this is the first time that we are doing a proof by induction. The main point to takeaway is that there is no magic...or in a sense, the only magic that we evoked is the Principle of Induction, which I stated without proof. Somehow it turns three pretty benign lemmas into a fact about all derivations! Then we applied the definition of TERM, and some basic reasoning by applying "foralls" and "if-thens" to relevant premises to get what we wanted. Now, most such proofs, when written by professional mathematicians are written in *much less detail*. By the time you are a professional, you don't want to dig through all of the details, you just want a sketch of the argument, and if some madman offered you your very own Maserati to write the full proof, you could fill in the details and drive away in a blaze of glory and the caustic smell of burnt rubber.

However, if you are a *digital computer*, in particular a mechanical proof checker, then you need way more detail than a professional mathematician to verify that a proof is true. A *theorem prover* or *mechanical proof assistant* can surely fill in some of these details (just like our functional program that automatically generates proofs of facts), but to be rock-solid sure that a theorem is true, there should be a pedantic *proof checker* hiding somewhere in your tool that takes the proof and simply checks that every last detail is there. This is how tools like the Coq proof assistant work: under the hood somewhere is the pedantic proof, waiting to be checked by a very simple, and pedantic, proof checker.

Later I'll show you what a mathematician's proof (which you can think of as pseudocode for a *real* proof, much like pseudocode for a real program) looks like. Those are nice for communicating with humans, but first you want to make sure that when push comes to shove, you can write the real thing, and understand that much of it is super-mechanical, and furthermore can be mechanically checked.

2.3 Deducing a New Induction Principle: Induction on Elements

Above we used induction to prove a universal fact about a particular function, exploiting its equational definition. Now let's use induction to construct a new general *proof principle*, which can streamline some other proofs that we will write. In math, as in CS we use our tools to build newer better tools!

Returning to the problems we set out at the start, we are interested in defining an evaluator over TERMS, which implies being able to reason about the TERMs in our language. However, we've inherited a principle for reasoning about *derivations*, not TERMs. Working with the derivations seems a bit indirect: notice all the work that we had to do above to get from a theorem about derivations to a theorem about terms. However, since we defined the set of TERMs using the derivations, one would expect that we could use this principle to prove properties of TERMs. Rather than fiddle with derivations every time we want to talk about TERMs, let's immediately use the principle of induction on derivations to establish that we can safely reason about TERMs directly.

Proposition 7 (Principle of Structural Induction on Elements $t \in \text{TERM}$)**.**
Let P be a predicate on TERMs *t. Then $P(t)$ holds for all* TERMs *t if:*

1. *$P(\textit{true})$ holds;*

2. *$P(\textit{false})$ holds;*

3. *If $P(t_1)$ $P(t_2)$, and $P(t_3)$ hold then $P(\textit{if}\ t_1\ \textit{then}\ t_2\ \textit{else}\ t_3)$ holds.*

Proof. Suppose we have a property P as described above. Then define a property Q of *derivations* as follows:

$$Q(\mathcal{D}) \equiv P(t) \text{ where } \mathcal{D} :: t \in \text{TERM}.$$

We can prove by Proposition 2 that $Q(\mathcal{D})$ holds for all derivations $\mathcal{D}$:

1. $Q\left(\overline{\mathsf{true} \in \text{TERM}}\right)$ holds since $P(\mathsf{true})$ holds;

2. $Q\left(\overline{\mathsf{false} \in \text{TERM}}\right)$ holds since $P(\mathsf{false})$ holds;

3. Suppose that $Q\left(\begin{array}{c}\mathcal{D}_1\\ r_1 \in \text{TERM}\end{array}\right)$, $Q\left(\begin{array}{c}\mathcal{D}_2\\ r_2 \in \text{TERM}\end{array}\right)$, and $Q\left(\begin{array}{c}\mathcal{D}_3\\ r_3 \in \text{TERM}\end{array}\right)$ hold for three derivations $\mathcal{D}_i$. Then by the definition of $Q(\mathcal{D})$ we know that $P(r_1)$, $P(r_2)$, and $P(r_3)$ must hold. By our assumptions

about P, it follows that $P(\text{if } r_1 \text{ then } r_2 \text{ else } r_3)$ holds. Then, appealing to our definition of Q, we conclude that

$$Q\left(\dfrac{\begin{array}{ccc}\mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_3\\ r_1 \in \textsc{Term} & r_2 \in \textsc{Term} & r_3 \in \textsc{Term}\end{array}}{\text{if } r_1 \text{ then } r_2 \text{ else } r_3 \in \textsc{Term}}\right)$$

holds.

Now appealing to the Principle of Structural Induction on Derivations, Q holds for all derivations $\mathcal{D}$. By definition of Q, this means that $P(t)$ holds for every term t with a derivation D, but remember that the set of terms is defined by the characteristic of having a derivation. Thus, $P(t)$ holds for all terms t. □

Proposition 7 *uses* the principle of induction on derivations to establish another theorem with the same general structure that lets us establish a property of all TERMs by proving a few small things. We'll see later in the course that there are many kinds of principles of induction, all of which fit a more general characterization, but for now these two kinds should get us pretty far.

A final note. Consider again Proposition 1, the principle of cases on derivations. We can *prove* it by induction on derivations $\mathcal{D}$, and in doing so, we *never* make use of any of the induction hypotheses. In that sense, proof by cases on derivations is a degenerate variant of proof by induction on derivations.

2.4 Induction on Propositional Entailment

To give you another example of induction on elements, as we are calling it, here is the statement of the Principle of Induction on Elements $\Gamma \vdash p$ **true** (i.e. elements $\langle \Gamma, p\rangle \in \cdot \vdash \cdot$ **true**). It is proven true in terms of the Principle of Induction on Derivations of $\mathcal{D} :: \Gamma \vdash p$ **true**, which we leave as an exercise for you to state. To keep it manageable, we will focus on just the fragment of propositional logic $p \in \textsc{Prop}$ corresponding to $p ::= A \mid \bot \mid p \wedge p \mid p \supset p$.

Proposition 8 (Principle of Induction on Elements of Entailment). *Let P be a property of entailments $\Gamma \vdash p$ **true**. Then P holds for all Γ, p such that $\Gamma \vdash p$ **true** if*

1. *$P(\Gamma \vdash p$ **true**$)$ holds whenever $p \in \Gamma$.*
2. *If $P(\Gamma \vdash \bot$ **true**$)$ holds then $P(\Gamma \vdash p$ **true**$)$ holds for all $p \in \textsc{Prop}$.*
3. *If $P(\Gamma \vdash p_1$ **true**$)$ and $P(\Gamma \vdash p_2$ **true**$)$ hold then $P(\Gamma \vdash p_1 \wedge p_2$ **true**$)$ holds.*
4. *If $P(\Gamma \vdash p_1 \wedge p_2$ **true**$)$ holds then $P(\Gamma \vdash p_1$ **true**$)$ holds and $P(\Gamma \vdash p_2$ **true**$)$ holds.*
5. *If $P(\Gamma \cup \{ p_1 \} \vdash p_2$ **true**$)$ holds then $P(\Gamma \vdash p_1 \supset p_2$ **true**$)$ holds.*
6. *If $P(\Gamma \vdash p_1 \supset p_2$ **true**$)$ and $P(\Gamma \vdash p_1$ **true**$)$ hold then $P(\Gamma \vdash p_2$ **true**$)$ holds.*

One side-note. Notice that Proposition 7 is said to be a principle of *structural induction* while Proposition 8 is not. That's because Proposition 7 follows the structure of the abstract syntax exactly: we just have to show that the property holds for an expression if it holds for each immediate subexpression. Similarly, Proposition 2 follows the structure of a derivation. On the other hand, the cases for Proposition 8 do not uniformly follow the structure of propositions p, sets of propositions Γ, or even pairs. Nonetheless the principle is justifiable. Later in the course we will talk about how we can come up with new induction principles for old sets.

I introduce this principle of induction primarily to show that we can easily get these principles, induction on derivations, and induction on elements, for *any* inductively defined set, regardless of what kind of set you are defining: syntax, relations, functions, or what have you.

3 Defining Functions

We've now developed a tool that we can use to prove properties of terms, but we have little experience using it yet, and furthermore we still need a way to justify our equational definitions of functions on terms. As it turns out, we can kill two birds with one stone: we will use Proposition 7 to establish a principle for defining functions over our infinite set of programs, and we'll use the result to produce our first non-trivial function over an inductively-defined infinite set. We'll also see that the structure of this function definition will allow use to reason about its properties. For instance, it will enable us to calculate how this function maps particular inputs to outputs.

Proposition 9 (Principle of Definition by Recursion on terms $t \in \text{TERM}$)**.** *Let S be a set and $s_t, s_f \in S$ be two elements and*

$$H_{if} : S \times S \times S \to S$$

be a function on S. Then there exists a unique function

$$F : \text{TERM} \to S$$

such that

1. *$F(\textit{true}) = s_t$;*
2. *$F(\textit{false}) = s_f$;*
3. *$F(\textit{if}\ t_1\ \textit{then}\ t_2\ \textit{else}\ t_3) = H_{if}(F(t_1), F(t_2), F(t_3))$.*

This principle can be proved using what we've already learned about inductive definitions and their associated induction principles. You'll get to see this in action on your homework.

Now, let's use Proposition 9 to define a function! According to the proposition all we need is:

1. some set (S);
2. two elements (s_t and s_f) of that set, though you can use the same element for both; and
3. some function (H_{if}) from any three elements of that set to a fourth.

For our example, I'll pick:

1. the set of natural numbers: $S = \mathbb{N}$.
2. the number 1 for s_t, and 0 for s_f: $s_t = 1$ and $s_f = 0$.
3. and the function $H(n_1, n_2, n_3) = n_1 + n_2 + n_3$ which just sums up all the numbers: $H : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

Well, then according to Proposition 7, there is a *unique* function $F : \text{TERM} \to \mathbb{N}$ with the properties that:

$$F(\text{true}) = 1; \tag{1}$$
$$F(\text{false}) = 0; \text{ and} \tag{2}$$
$$F(\text{if}\ t_1\ \text{then}\ t_2\ \text{else}\ t_3) = H(F(t_1), F(t_2), F(t_3)) = F(t_1) + F(t_2) + F(t_3). \tag{3}$$

That means that these three equations (properties) *uniquely* characterize some function from TERMs to functions. At this point all we know is that there *is* a function that satisfies these properties, and that there's only one. It's a black box to us except for these equations, which tell us something about the function. We could learn some things about this function by using the equations to calculate what the function maps certain terms to, but I'll save that exercise for homework. Instead, I hope you'll trust me that this function maps every TERM in our language to the number of trues in it. We can call this function $trues$.

What I've shown here is a rather *longhand* way of writing down a function definition: we take the Principle of Recursion at its word literally, choose the necessary components, and then conclude that there's some function that satisfies the set of equations that you get after you specialize the proposition for your particular choices (as I've done above). In textbooks and papers, writers rarely show things in this much painful detail. Instead, they cut to the chase and simply give the equations that you get at the end. We'll call that the *shorthand* way of defining a function by recursion.[5]

Considering our example again, here is the typical shorthand definition of the same function:

Definition 2. *The function* $trues : \textsc{Term} \to \mathbb{N}$ *is defined by*

$$\begin{aligned} trues(\textsf{true}) &= 1 \\ trues(\textsf{false}) &= 0 \\ trues(\textsf{if}\, t_1 \,\textsf{then}\, t_2 \,\textsf{else}\, t_3) &= trues(t_1) + trues(t_2) + trues(t_3). \end{aligned}$$

The function is given by the final specialized version of the equations, and it's up to you (the reader) to figure out which S, s_t, s_f and H_{if} give you these equations ... which is often not hard.

The shorthand definition above can be read this way: ignoring the name we're giving the function ($trues$), and simply taking the three equations, we are saying that:

$$trues \in \{ F \in \textsc{Term} \to \mathbb{N} \mid P(F) \}$$

Where $P(F)$ is the combination of equations (1)-(3) above. The important thing is that in order for $P(F)$ to be a good definition, the set we define above should have *exactly* one element, which means that $trues$ is that element.

Why am I beating this to death? Because it's easy to write a so-called "definition" with equations that's not a definition at all![6] Let's consider two simple examples. Take the natural numbers $\mathbb{N}$, and suppose I claim I'm defining a function $F : \mathbb{N} \to \mathbb{N}$ by the equation $F(n) = F(n)$. Well that doesn't define a *unique* function at all because

$$\{ F \in \mathbb{N} \to \mathbb{N} \mid \forall n \in N.F(n) = F(n) \} = \mathbb{N} \to \mathbb{N} \text{ !!!}$$

That is to say, our equation picks *all* of the functions, not one. This is fine if you are specifically picking a class of functions and you don't care which one it is, but you'd better know that you're not naming a single function! This is not a function definition because it picks too many functions.

For a second broken example, suppose our equation is $F(n) = 1 + F(n)$. This one is broken for the opposite reason:

$$\{ F \in \mathbb{N} \to \mathbb{N} \mid \forall n \in N.F(n) = 1 + F(n) \} = \emptyset!$$

There are *no* functions with this property. So this is not a definition because it picks too few functions. Definitions like this are particularly bad because we can prove all sorts of terribly wrong things by taking reasoning steps that use a function that *doesn't exist*! Here's a fun example of what can go wrong:

$$\begin{aligned} 1 &= 1 && \text{by identity;} \\ F(1) &= F(1) && \text{by applying a function to two equals;} \\ F(1) &= 1 + F(1) && \text{by the "definition" of F;} \\ F(1) - F(1) &= 1 + F(1) - F(1) && \text{by subtracting equals from equals;} \\ 0 &= 1 && \text{by simple arithmetic .} \end{aligned}$$

So by reasoning with a function that we could never have, we prove something that's totally, albeit obviously, false. In this case we can see that something went terribly wrong, but what's really bad is when you "prove" something that isn't obviously false, but is false nevertheless. Granted the example above is harebrained, but plenty of prospective theorists have been led astray by morally doing exactly this kind of thing, and then thinking that they have proven an interesting theorem when in fact they have done no such thing because they started with a bad definition.

The point is that when you try to define a function (or other single elements) by providing a set of properties, you are obliged to show that those properties *uniquely* characterize the function. The Principle of Recursion is a great workhorse because it once-and-for-all dispatches that obligation for those sets of equations that can be stated in the form that it discusses: as long as our equations fit the Principle of Recursion,

[5] This *shorthand* and *longhand* terminology is my own creation. I don't think you'll find it in the literature.
[6] Sadly I see it in research papers (and textbooks) all too often!

we know that we have a real function definition. Now, whether the function you have successfully defined is the one that you really wanted is a different, more philosophically interesting question. For example, how would you go about arguing that the $trues$ function *really does* count the number of trues in a term? That one isn't too bad, but in general, arguing that your formalization of a previously vague and squishy concept is "the right definition" is a matter of analytic philosophy, a rather challenging field of inquiry.

4 A Small Case Study

For more experience, let's revisit our $bools$ function, which I simply claimed was properly defined. First let's see the *shorthand* style again:

Definition 3. *Let* $bools : \text{TERM} \to \mathbb{N}$ *be defined by*

$$\begin{aligned} bools(\textsf{true}) &= 1 \\ bools(\textsf{false}) &= 1 \\ bools(\textsf{if}\ t_1\ \textsf{then}\ t_2\ \textsf{else}\ t_3) &= bools(t_1) + bools(t_2) + bools(t_3) \end{aligned}$$

Based on the material up above, we can unpack this definition into the low-level pieces that correspond to the principle of recursion. Here is the *longhand* presentation:

1. $S = \mathbb{N}$;
2. $s_t = 1$, $s_f = 1$
3. $H_{if} : \mathbb{N} \times \mathbb{N} \times \mathbb{N}; Hif(n_1, n_2, n_3) = n_1 + n_2 + n_3$.

You should convince yourself that this function yields the number of falses and trues in a TERM. At this point, the only way that you can do that is to use the function to reason about enough examples that you have confidence that your function meets your "informal specification." This is pretty much like programming practice: you need enough "unit tests" to convince yourself and others that you have the right specification. All we had done so far is prove that what we have in our hands is a *proper specification* of *some specific function*: whether it gives you what you want or not is a totally separate question!

A mathematician's proof We now have *two* principles for proving hard facts about TERMs, and we have a principle for defining functions on TERMs. To wrap things up, let's revisit the proof that $bools(t) > 0$. Here is the way that a mathematician would write that proof, at least using our shiny new principle of induction on elements $t \in \text{TERM}$. Here is a statement and a proof of that statement as you would likely see it in a textbook:

Proposition 10. $bools(t) > 0$ *for all* $t \in \text{TERM}$

Proof. By induction on t

Case (true). $bools(\textsf{true}) = 1 > 0$

Case (false). Analogous to true.

Case (if). If $bools$ yields a positive number for each subcomponent of if then their sum will be positive too. □

Wow, so shiny and tiny! Seeing the relation between the above conversational statements and the precise pedantic formal principal of induction that we presented above time may not be all that obvious at first, but if you start from the principle above, you should be able to figure out a formal property P and recast each of the cases as one of the pieces of the statement of the principle of induction. The form you see here is typical of what shows up in the literature. It's important to be able to make that connection if you hope to really understand proofs and be able to check whether they are correct. Going forward, you will see more examples.

To better understand the connection, I recommend that you rewrite the above proof in the more precise (longhand) style to ensure that you can.

5 Parting Thoughts

To wrap up, the last couple of classes we have been addressing two issues. We have been introducing some preliminary notions from the semantics of programming languages, and at the same time establishing a common understanding for how the math underlying those semantics "works."

On the semantics front, we've talked about the idea of a language being defined as some set of programs (the "syntax" if you will), and a mapping from programs to observable results (the "semantics"). Our examples have been simple so far, but we've observed that whatever approach we use to define the evaluator has a significant impact on how we reason about our language and its programs.

On the mathematical front, we discussed some of the basic ways of building sets:

1. enumerating elements, which works for a finite set (e.g., $\{1, 2\}$): Along with it comes reasonign by cases;

2. taking the union ($A \cup B$) or intersection ($A \cap B$) of sets that you already have (A and B): for these we reason by disjunction ("or") or conjunction ("and");

3. forming the product ($A \times B$) of two sets.

4. Using separation to filter the elements of some set $\{ a \in A \mid P(a) \}$ according to some predicate P. This too gives us a proof principle.

Inductive definitions with rules and definition of functions by (recursive) equations are simply particular instances of item (4) above, where the elements are filtered based on the existence of derivations and the satisfaction of those equations, respectively.

At this juncture we have enough mathematical machinery to draw our focus more on the programming language concepts. Any new mathematical concepts we need can be weaved in on demand.

References

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.

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`.

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