Lecture Notes on Natural Numbers

15-317: Constructive Logic
Frank Pfenning[*]

Lecture 6
September 20, 2016

1 Introduction

In this lecture we discuss the data type of natural numbers. They serve as a prototype for a variety of inductively defined data types, such as lists or trees. Together with quantification as introduced in the previous lecture, this allow us to reason constructively about natural numbers and extract corresponding functions. The constructive system for reasoning logically about natural numbers is called *intuitionistic arithmetic* or *Heyting arithmetic*.

2 Induction

As usual, we think of the type of natural numbers as defined by its introduction form. Note, however, that nat is a *type* rather than a proposition. It is possible to completely unify these concepts to arrive at *type theory*, something we might explore later in this course. For now, we just specify cases for the typing judgment $t : \tau$, read term t has type τ, that was introduced in the previous lecture on quantification, but for which we have seen no specific instances yet. We distinguish this from $M : A$ which has the same syntax, but relates a proof term to a proposition instead of a term to a type.

There are two introduction rules, one for zero and one for successor.

$$\frac{}{\mathsf{0} : \mathsf{nat}}\ \mathsf{nat}I_0 \qquad \frac{n : \mathsf{nat}}{\mathsf{s}\,n : \mathsf{nat}}\ \mathsf{nat}I_s$$

[*]Edits by André Platzer.

Intuitively, these rules express that 0 is a natural number ($\mathsf{nat}I_0$) and that the successor $\mathsf{s}\,n$ is a natural number if n is a natural number. This definition has a different character from the previous definitions. For example, we defined the meaning of $A \wedge B$ *true* from the meanings of A *true* and the meaning of B *true*, all of which are propositions. It is even different from the proof term assignment rules where, for example, we defined $\langle M, N\rangle : A \wedge B$ in terms of $M : A$ and $N : B$. In each case, the proposition is decomposed into its parts.

Here, the types in the conclusion and premise of the $\mathsf{nat}I_s$ rules are the same, namely nat. Fortunately, the *term* n in the premise is a part of the term $\mathsf{s}\,n$ in the conclusion, so the definition is not circular, because the judgment in the premise is still smaller than the judgment in the conclusion. In (verificationist) constructive logic truth is defined by the introduction rules. The resulting implicit principle, that nothing is true unless the introduction rules prove it to be true, is of deep significance here. Nothing else is a natural number, except the objects constructed via $\mathsf{nat}I_s$ from $\mathsf{nat}I_0$. The rational number $\frac{7}{4}$ cannot sneak in claiming to be a natural number (which, by $\mathsf{nat}I_s$ would also make its successor $\frac{11}{4}$ claim to be natural).

But what should the elimination rule be? We cannot decompose the proposition into its parts, so we decompose the term instead. Natural numbers have two introduction rules just like disjunctions. Their elimination rule, thus, also proceeds by cases, accounting for the possibility that a given n of type nat is either $\mathsf{0}$ or $\mathsf{s}\,x$ for some x. A property $C(n)$ is true if it holds no matter whether the natural number n was introduced by $\mathsf{nat}I_0$ so is zero or was introduced by $\mathsf{nat}I_s$ so is a successor.

$$\dfrac{n : \mathsf{nat} \quad C(\mathsf{0})\ \mathit{true} \quad \begin{array}{c} \overline{x : \mathsf{nat}} \quad \overline{C(x)\ \mathit{true}}^{\,u} \\ \vdots \\ C(\mathsf{s}\,x)\ \mathit{true}\end{array}}{C(n)\ \mathit{true}}\ \mathsf{nat}E^{x,u}$$

In words: In order to prove property C of a natural number n we have to prove $C(\mathsf{0})$ and also $C(\mathsf{s}\,x)$ under the assumption that $C(x)$ for a new parameter x. The scope of x and u is just the rightmost premise of the rule. This corresponds exactly to proof by induction, where the proof of $C(\mathsf{0})$ is the base case, and the proof of $C(\mathsf{s}\,x)$ from the assumption $C(x)$ is the induction step. That is why $\mathsf{nat}E^{x,u}$ is also called an *induction rule* for nat.

We managed to state this rule without any explicit appeal to universal quantification, using parametric judgments instead. We could, however, write it down with explicit quantification, in which case it becomes:

$$\forall n{:}\mathsf{nat}.\ C(\mathsf{0}) \supset (\forall x{:}\mathsf{nat}.\ C(x) \supset C(\mathsf{s}\,x)) \supset C(n)$$

for an arbitrary property C of natural numbers. It is an easy exercise to prove this with the induction rule above, since the respective introduction rules lead to a proof that exactly has the shape of $\mathsf{nat}E^{x,u}$.

All natural numbers are zeros or successors. To illustrate this rule in action, we start with a very simple property: every natural number is either $\mathsf{0}$ or has a predecessor. First, a detailed induction proof in the usual mathematical style and then a similar formal proof.

Theorem: $\forall x{:}\mathsf{nat}.\ x = \mathsf{0} \lor \exists y{:}\mathsf{nat}.\ x = \mathsf{s}\,y$.
Proof: By induction on x.

Case: $x = \mathsf{0}$. Then the left disjunct is true.

Case: $x = \mathsf{s}\,x'$. Then the right disjunct is true: pick $y = x'$ and observe $x = \mathsf{s}\,x' = \mathsf{s}\,y$.

Next we write this in the formal notation of inference rules. We suggest the reader try to construct this proof step-by-step; we show only the final deduction. We assume there is either a primitive or derived rule of inference called refl expressing reflexivity of equality on natural numbers ($n = n$). We use the same names as in the mathematical proof.

$$\dfrac{\dfrac{\dfrac{}{x : \mathsf{nat}} \quad \dfrac{\dfrac{}{\mathsf{0} = \mathsf{0}\ \mathit{true}}\,\mathsf{refl}}{\mathsf{0} = \mathsf{0} \lor \exists y{:}\mathsf{nat}.\ \mathsf{0} = \mathsf{s}\,y\ \mathit{true}}\,{\lor}I_1 \quad \dfrac{\dfrac{\dfrac{}{x' : \mathsf{nat}} \quad \dfrac{}{\mathsf{s}\,x' = \mathsf{s}\,x'\ \mathit{true}}\,\mathsf{refl}}{\exists y{:}\mathsf{nat}.\ \mathsf{s}\,x' = \mathsf{s}\,y\ \mathit{true}}\,\exists I}{\mathsf{s}\,x' = \mathsf{0} \lor \exists y{:}\mathsf{nat}.\ \mathsf{s}\,x' = \mathsf{s}\,y\ \mathit{true}}\,{\lor}I_2}{x = \mathsf{0} \lor \exists y{:}\mathsf{nat}.\ x = \mathsf{s}\,y\ \mathit{true}}\,\mathsf{nat}E^{x',u}}{\forall x{:}\mathsf{nat}.\ x = \mathsf{0} \lor \exists y{:}\mathsf{nat}.\ x = \mathsf{s}\,y\ \mathit{true}}\,\forall I^x$$

This is a simple proof by cases and, in this particular proof, does not even use the induction hypothesis $x' = \mathsf{0} \lor \exists y{:}\mathsf{nat}.\ x' = \mathsf{s}\,y\ \mathit{true}$, which would have been labeled u. It is also possible to finish the proof by eliminating from that induction hypothesis, but the proof then ends up being more complicated. At our present level of understanding, the computational counterpart for the above proof might be a zero-check function for natural numbers. It takes any natural number and provides the left disjunct if that number was 0 while providing the right disjunct if it was a successor. Making use of the witness, we will later discover more general computational content once we have a proof term assignment.

In the application of the induction rule $\mathsf{nat}E$ we used the property $C(x)$, which is a proposition with the free variable x of type nat. To write it out explicitly:

$$C(x) = (x = \mathsf{0} \vee \exists y{:}\mathsf{nat}.\ x = \mathsf{s}\,y)$$

While getting familiar with formal induction proofs it may be a good idea to write out the induction formula explicitly.

Odds or evens. As a second example we specify a function which tests if its argument is even or odd. For this purpose we assume a doubling function $2 \times$. Equality is decided by an oracle.

> **Theorem:** $\forall x{:}\mathsf{nat}.\ (\exists y.\ x = 2 \times y) \vee (\exists z.\ x = \mathsf{s}\,(2 \times z))$.
> **Proof:** By induction on x.

Case: $x = \mathsf{0}$. Then pick $y = \mathsf{0}$ since $\mathsf{0} = 2 \times \mathsf{0}$.

Case: $x = \mathsf{s}\,x'$. By induction hypothesis we have either $\exists y.\ x' = 2 \times y$ or $\exists z.\ x' = \mathsf{s}\,(2 \times z)$. We distinguish these two cases.

- Case: $\exists y.\ x' = 2 \times y$. Then the second disjunct holds because we can pick $z = y$: $x = \mathsf{s}\,x' = \mathsf{s}\,(2 \times y)$.
- Case: $\exists z.\ x' = \mathsf{s}\,(2 \times z)$. Then the first disjunct holds because we can pick $y = \mathsf{s}\,z$: $x = \mathsf{s}\,x' = \mathsf{s}\,(\mathsf{s}\,(2 \times z)) = 2 \times (\mathsf{s}\,z)$ by properties of $2 \times$.

We start the transcription of this proof, but write A instead of A *true* for space reasons.

$$\dfrac{\dfrac{\dfrac{}{x : \mathsf{nat}} \quad \dfrac{\mathsf{0} = 2 \times \mathsf{0} \vee \ldots}{(\exists y.\ \mathsf{0} = 2 \times y) \vee \ldots}\,\exists I \quad \begin{array}{c} \dfrac{}{x' : \mathsf{nat}} \quad \dfrac{}{(\exists y.\ x' = 2 \times y) \vee (\exists z.\ x' = \mathsf{s}\,(2 \times z))}\,u \\ \vdots \\ (\exists y.\ \mathsf{s}\,x' = 2 \times y) \vee (\exists z.\ \mathsf{s}\,x' = \mathsf{s}\,(2 \times z)) \end{array}}{(\exists y.\ x = 2 \times y) \vee (\exists z.\ x = \mathsf{s}\,(2 \times z))}\,\mathsf{nat}E^{x',u}}{\forall x{:}\mathsf{nat}.\ (\exists y.\ x = 2 \times y) \vee (\exists z.\ x = \mathsf{s}\,(2 \times z))}\,\forall I^{x}$$

From here, we proceed by an $\vee E$ applied to u, followed by an $\exists E$ in each branch, naming the y and z that are known to exist. Unfortunately, the 2-dimensional notation for natural deductions which is nice and direct for describing and reasoning about the rules, is not so good for writing actual formal deductions.

3 Local Proof Reduction

We check that the rules are locally sound and complete. For soundness, we verify that no matter how we introduce the judgment $n : \mathsf{nat}$, we can find a more direct proof of the conclusion. In the case of $\mathsf{nat}I_0$ this is easy to see, because the second premise already establishes our conclusion directly.

$$\dfrac{\dfrac{}{\mathsf{0} : nat}\,\mathsf{nat}I_0 \qquad \begin{array}{c}\mathcal{E}\\ C(\mathsf{0})\ true\end{array} \qquad \begin{array}{c}\overline{x : nat} \quad \overline{C(x)\ true}^{\,u}\\ \mathcal{F}\\ C(\mathsf{s}\,x)\ true\end{array}}{C(\mathsf{0})\ true}\,\mathsf{nat}E^{x,u} \quad \Longrightarrow_R \quad \begin{array}{c}\mathcal{E}\\ C(\mathsf{0})\ true\end{array}$$

The case where $n = \mathsf{s}\,n'$ is more difficult and more subtle. Intuitively, we should be using the deduction of the second premise for this case.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ n' : \mathsf{nat}\end{array}}{\mathsf{s}\,n' : \mathsf{nat}}\,\mathsf{nat}I_s \qquad \begin{array}{c}\mathcal{E}\\ C(\mathsf{0})\ true\end{array} \qquad \begin{array}{c}\overline{x : \mathsf{nat}} \quad \overline{C(x)\ true}^{\,u}\\ \mathcal{F}\\ C(\mathsf{s}\,x)\ true\end{array}}{C(\mathsf{s}\,n')\ true}\,\mathsf{nat}E^{x,u}$$

$$\Longrightarrow_R \quad \begin{array}{c} \begin{array}{c}\mathcal{D}\\ n' : \mathsf{nat}\end{array} \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ n' : nat\end{array} \quad \begin{array}{c}\mathcal{E}\\ C(\mathsf{0})\ true\end{array} \qquad \begin{array}{c}\overline{x : \mathsf{nat}} \quad \overline{C(x)\ true}^{\,u}\\ \mathcal{F}\\ C(s\,x)\ true\end{array}}{C(n')\ true}\,\mathsf{nat}E^{x,u} \\ [n'/x]\mathcal{F}' \\ C(s\,n')\ true \end{array}$$

It is difficult to see in which way this is a reduction: $\mathcal{D}$ is duplicated, $\mathcal{E}$ persists, and we still have an application of $\mathsf{nat}E$. The key is that the term we are eliminating with the applicaton of $\mathsf{nat}E$ becomes smaller: from $\mathsf{s}\,n'$ to n'. In hindsight we should have expected this, because the term is also the only component getting smaller in the second introduction rule for natural numbers. Fortunately, the term that $\mathsf{nat}E$ is applied to can only get smaller finitely often because it will ultimately just be 0, so will be back in the first local reduction case.

The computational content of this reduction is more easily seen in a different context, so we move on to discuss primitive recursion.

The question of local expansion does not make sense in our setting. The difficulty is that we need to show that we can apply the elimination rules in such a way that we can reconstitute a proof of the original judgment. However, the elimination rule we have so far works only for the truth judgment, so we cannot really reintroduce $n : \mathsf{nat}$, since the only two introduction rules $\mathsf{nat}I_0$ and $\mathsf{nat}I_s$ do not apply. The next section will give us the tool.

4 Primitive Recursion

Reconsidering the elimination rule for natural numbers, we can notice that we exploit the knowledge that $n : \mathsf{nat}$, but we only do so when we are trying to establish the truth of a proposition, $C(n)$. However, we are equally justified in using $n : \mathsf{nat}$ when we are trying to establish a typing judgment of the form $t : \tau$. The rule, also called *recursion rule* for nat, then becomes

$$\dfrac{n : \mathsf{nat} \quad t_0 : \tau \quad \begin{array}{c} \overline{x : \mathsf{nat}} \quad \overline{r : \tau} \\ \vdots \\ t_s : \tau \end{array}}{R(n, t_0, x.\, r.\, t_s) : \tau}\ \mathsf{nat}E^{x,r}$$

Here, R is a new term constructor,[1] the term t_0 is the zero case where $n = 0$, and the term t_s captures the successor case where $n = \mathsf{s}\, n'$. In the latter case x is a new parameter introduced in the rule that stands for n'. And r stands for the result of the function R when applied to n', which corresponds to an appeal to the induction hypothesis. Recall that the dot notation $x.\, r.\, t_s$ is a notational reminder that occurrences of x and r in t_s are bound. Of course the fact that both are bound correspond to the assumptions $x : \mathsf{nat}$ and $r : \tau$ that are introduced to prove $t_s : \tau$ in the rightmost premise.

The local reduction rules may help explain this. We first write then down just on the terms, where they are computation rules.

$$\begin{array}{lll} R(\mathsf{0}, t_0, x.\, r.\, t_s) & \Longrightarrow_R & t_0 \\ R(\mathsf{s}\, n', t_0, x.\, r.\, t_s) & \Longrightarrow_R & [R(n', t_0, x.\, r.\, t_s)/r][n'/x]\, t_s \end{array}$$

[1] R suggests recursion

The second case reduces to the term t_s with parameter x instantiated to the number n' of the inductive hypothesis and parameter r instantiated to to the value of R at n'. So the argument t_0 of R indicates the output to use for $n = \mathsf{0}$ and t_s indicates the output to use for $n = \mathsf{s}x$ as a function of the smaller number x and of r for the recursive outcome of $R(n, t_0, x.\, r.\, t_s)$.

These are still quite unwieldy, so we consider a more readable schematic form, called the *schema of primitive recursion*. If we define f by cases

$$\begin{array}{lcl} f(\mathsf{0}) & = & t_0 \\ f(s\,x) & = & t_s(x, f(x)) \end{array}$$

where the only occurence of f on the right-hand side is applied to x, then we could have defined f explicitly with

$$f = \lambda n.\, R(n, t_0, x.\, r.\, t_s(x, r)).$$

To verify this, apply f to $\mathsf{0}$ and apply the reduction rules and also apply f to $\mathsf{s}\,n$ for an arbitrary n and once again apply the reduction rules.

$$\begin{array}{lcl} f(\mathsf{0}) & \Longrightarrow_R & R(\mathsf{0}, t_0, x.\, r.\, t_s(x, r)) \\ & \Longrightarrow_R & t_0 \end{array}$$

noting that the x in $x.r.t_s(\ldots)$ is not a free occurrence (indicated by the presence of the dot in x.) since it corresponds to the hypothesis $x : \mathsf{nat}$ in $\mathsf{nat}E^{x,r}$. Finally

$$\begin{array}{lcl} f(\mathsf{s}\,n) & \Longrightarrow_R & R(\mathsf{s}\,n, t_0, x.\, r.\, t_s(x, r)) \\ & \Longrightarrow_R & t_s(n, R(n, t_0, x.\, r.\, t_s(x, r))) \\ & = & t_s(n, f(n)) \end{array}$$

The last equality is justified by a (meta-level) induction hypothesis, because we are trying to show that $f(n) = R(n, t_0, x.\, r.\, t_s(x, r))$

In Proofs as Programs we saw λ-abstraction of the form $\lambda u{:}A.\ M$ for a proposition A. Here we are using λ-abstraction on natural number data. To be completely formal, we would also have to define the function space on data, which comes from the following pair of introduction and elimination rules for $\tau \to \sigma$. Since they are completely analogous to implication, except for using terms instead of proof terms, we will not discuss them further

$$\dfrac{\begin{array}{c}\overline{x : \tau} \\ \vdots \\ s : \sigma\end{array}}{\lambda x{:}\tau.\ s : \tau \to \sigma} \to\! I \qquad \dfrac{s : \tau \to \sigma \quad t : \tau}{s\,t : \sigma} \to\! E$$

The local reduction is

$$(\lambda x{:}\tau.\ s)\,t \quad \Longrightarrow_R \quad [t/x]s$$

Now we can define double via the schema of primitive recursion.

$$\begin{array}{lcl} \mathsf{double}(0) & = & 0 \\ \mathsf{double}(\mathsf{s}\,x) & = & \mathsf{s}\,(\mathsf{s}\,(\mathsf{double}\,x)) \end{array}$$

We can read off the closed-form definition if we wish:

$$\mathsf{double} = \lambda n.\ R(n, 0, x.\,r.\,\mathsf{s}\,(\mathsf{s}\,r))$$

After having understood this, we will be content with using the schema of primitive recursion. We define addition and multiplication as exercises.

$$\begin{array}{lcl} \mathsf{plus}(0) & = & \lambda y.\ y \\ \mathsf{plus}(\mathsf{s}\,x) & = & \lambda y.\ \mathsf{s}\,((\mathsf{plus}\,x)\,y) \end{array}$$

Notice that plus is a function of type $\mathsf{nat} \to (\mathsf{nat} \to \mathsf{nat})$ that is primitive recursive in its (first and only) argument.

5 Proof Terms

With proof terms for primitive recursion in place, we can revisit and make a consistent proof term assignment for the elimination form with respect to the truth of propositions.

$$\dfrac{n : \mathsf{nat} \quad M_0 : C(0)\ \mathit{true} \qquad \begin{array}{c} \overline{x : \mathsf{nat}} \quad \overline{u : C(x)\ \mathit{true}}^{\,u} \\ \vdots \\ M_s : C(\mathsf{s}\,x)\ \mathit{true} \end{array}}{R(n, M_0, x.\,u.\,M_s) : C(n)\ \mathit{true}}\ \mathsf{nat}E^{x,u}$$

The local reductions we discussed before for terms representing data, also work for these proofs terms, because they are both derived from slightly different variants of the elimination rules (one with proof terms, one with data terms).

$$\begin{array}{lcl} R(0, M_0, x.\,u.\,M_s) & \Longrightarrow_R & M_0 \\ R(\mathsf{s}\,n', M_0, x.\,u.\,M_s) & \Longrightarrow_R & [R(n', M_0, x.\,u.\,M_s)/u][n'/x]\,M_s \end{array}$$

We can conclude that proofs by induction correspond to functions defined by primitive recursion, and that they compute in the same way.

Returning to the earlier example, we can now write the proof terms, using $\_$ for proofs of equality (whose computational content we do not care about).

Theorem: $\forall x{:}\mathsf{nat}.\ x = 0 \lor \exists y{:}\mathsf{nat}.\ x = \mathsf{s}\,y$.
Proof: By induction on x.

Case: $x = 0$. Then the left disjunct is true.

Case: $x = \mathsf{s}\,x'$. Then the right disjunct is true: pick $y = x'$ and observe $x = \mathsf{s}\,x' = \mathsf{s}\,y$.

The extracted function is the predecessor function:

$$\mathsf{pred} \quad = \quad \lambda x{:}\mathsf{nat}.\ R(x, \mathbf{inl}\,\_, x.\,r.\,\mathbf{inr}\langle x, \_\rangle)$$

More easily readable is the ML version, where we have eliminated the computationally irrelevant parts from above.

```
datatype nat = Z | S of nat;
datatype nat_option = Inl | Inr of nat
(* pred : nat -> nat_option *)
fun pred Z = Inl
  | pred (S x) = Inr x
```

Similarly, recall the proof of the even-or-odd theorem:

Theorem: $\forall x{:}\mathsf{nat}.\ (\exists y.\ x = 2 \times y) \lor (\exists z.\ x = \mathsf{s}\,(2 \times z))$.
Proof: By induction on x.

Case: $x = 0$. Then pick $y = 0$ since $0 = 2 \times 0$.

Case: $x = \mathsf{s}\,x'$. By induction hypothesis we have either $\exists y.\ x' = 2 \times y$ or $\exists z.\ x' = \mathsf{s}\,(2 \times z)$. We distinguish these two cases.

Case: $\exists y.\ x' = 2 \times y$. Then the second disjunct holds because we can pick $z = y$: $x = \mathsf{s}\,x' = \mathsf{s}\,(2 \times y)$.

Case: $\exists z.\ x' = \mathsf{s}\,(2 \times z)$. Then the first disjunct holds because we can pick $y = \mathsf{s}\,z$: $x = \mathsf{s}\,x' = \mathsf{s}\,(\mathsf{s}\,(2 \times z)) = 2 \times (\mathsf{s}\,z)$ by properties of $2 \times$.

The extracted function returns half of its input along with an indication whether that is an even or an odd number, corresponding to the two disjuncts:

$$\begin{array}{rcl} \mathsf{half} & = & \lambda x{:}\mathsf{nat}.\ R(x, \mathbf{inl}\langle 0, \_\rangle, \\ & & \quad x.\,r.\,\mathbf{case}\ r\ \ \mathbf{of}\ \ \mathbf{inl}\,u \Rightarrow \mathbf{let}\ \langle y, \_\rangle = u\ \ \mathbf{in}\ \ \mathbf{inr}\,\langle y, \_\rangle \\ & & \qquad\qquad\qquad \mid \mathbf{inr}\,w \Rightarrow \mathbf{let}\ \langle z, \_\rangle = w\ \ \mathbf{in}\ \ \mathbf{inl}\,\langle \mathsf{s}\,z, \_\rangle \end{array}$$

or, in ML, where `half(2` $\times$ n`)` returns `Even(`n`)` and `half(2` $\times$ $n + 1$`)` returns `Odd(`n`)`.

```
datatype nat = Z | S of nat;
datatype parity = Even of nat | Odd of nat
(* half : nat -> parity *)
fun half Z = Even Z
  | half (S x) = (case half x
                    of Even y => Odd y
                     | Odd z => Even (S z))
```

6 Local Expansion

Using primitive recursion, we can now write a local expansion.

$$\begin{array}{c} \mathcal{D} \\ n : \mathsf{nat} \end{array} \quad \Longrightarrow_E \quad \dfrac{\begin{array}{c} \mathcal{D} \\ n : \mathsf{nat} \end{array} \quad \dfrac{}{0 : \mathsf{nat}}\,\mathsf{nat}I_0 \quad \dfrac{\overline{x : \mathsf{nat}}}{\mathsf{s}\,x : \mathsf{nat}}\,\mathsf{nat}I_s}{R(n, 0, x.\,r.\,\mathsf{s}\,x) : \mathsf{nat}}\,\mathsf{nat}E^{x,r}$$

A surprising observation about the local expansion is that it does not use the recursive result, r, which corresponds to a use of the induction hypothesis. Consequently, a simple proof-by-cases that uses $\mathsf{nat}E_0$ when n is zeroand uses $\mathsf{nat}E_s$ when n is a successor would also have been locally sound and complete.

This is a reflection of the fact that the local completeness property we have does not carry over to a comparable global completeness. The difficulty is the well-known property that in order to prove a proposition A by induction, we may have to first generalize the induction hypothesis to some B, prove B by induction and also prove $B \supset A$. Such proofs do not have the subformula property, which means that our strict program of explaining the meaning of propositions from the meaning of their parts breaks down in arithmetic. In fact, there is a hierarchy of arithmetic theories, depending on which propositions we may use as induction formulas.