Country-Fried Coq: Overly-Formalized Nonstandard Arithmetic

with Applications to Computer Science Research

Michael Coulombe

Abstract

Country-Fried Coq is an extension to the popular proof assistant software Coq to support nonstandard arithmetic. Did you know there is a natural number k with the properties "0 < k", "1 < k", "2 < k", "3 < k", and so on for EVERY numeral? In any nonstandard proof using axioms { "n_1 < k", "n_2 < k , ... , "n_m < k" }, we could just redefine k as max(n_1, n_2, ... , n_m)+1 to get a standard proof. In this paper, I give a novel approach for taking way too long to formally-verify this simple idea in Coq; introduce Country-Fried Coq, which adds this number to Coq itself; and show that it has a home in your computer science recipe book.

Keywords Formal Verification, Peano Arithmetic, Nonstandard Methods, Fried Foods

1. Introduction

The natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, \ldots\}$ are very important, especially in computer science. Peano Arithmetic (PA) is a theory that tried to outline the defining properties of these numbers, along with operators $+$ and $\times$, in formal logic [8]. However, mathematicians have come to show that the familiar $\mathbb{N}$ is not the only model of PA: other models satisfy all the axioms, like $\mathbb{N}^*$ shown in Figure 1.1. These nonstandard models contain "infinite-sized" nonstandard natural numbers, giving rise to nonstandard PA theories with extra axioms to define their properties, as well as theories of nonstandard real numbers and more.

Coq is a widely-used proof assistant software based on constructive type theory. It tricks you into thinking you are writing and verifying formal proofs, but it's actually just functional programming with extra features. No background is required or provided.

In this paper, I give a new implementation of PA and nonstandard PA in Coq; prove how to transfer proofs between both theories and turn them into Coq proofs about the `nat` type, verifying their consistency; invent a new transfer method; describe Country-Fried Coq, which makes Coq itself nonstandard; and explain why computer scientists should care. Buckle up and enjoy the ride!

SIGTBD 2022 April 1st, 2022, Cambridge, Massachusetts, USA.

DOI: http://dx.doi.org/10.1145/(to come)

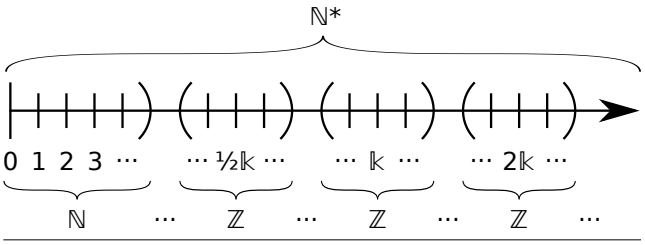

Figure 1.1. The Nonstandard Natural Number Line. The model $\mathbb{N}^*$ has a juicy and tender initial segment of all standard naturals $\mathbb{N}$ covered in a crisp, buttermilky breading made with infinitely-many copies of the integers $\mathbb{Z}$, as dense as the rationals $\mathbb{Q}$.

2. Representing Peano Arithmetic in Coq

2.1 Language of Arithmetic

The language of PA is made up of *expressions* and *propositions*. An expression is represented with the `PeanoExpr` data type, and a proposition is represented with the `PeanoProp` data type.

(N)`PeanoExpr`	Meaning
0	Zero
Sx	Successor of x
$x + y$	Addition of x and y
$x \times y$	Multiplication of x and y
v_i	Variable with index i: `nat`
$\mathbb{k}$	The nonstandard number $\mathbb{k} > \mathbb{N}$ (only present in `NPeanoExpr`)

(N)`PeanoProp`	Meaning
$\top$	True
$\bot$	False
$\neg p$	Not p
$x = y$	(N)`PeanoExpr` x equals y
$p \vee q$	p or q
$p \wedge q$	p and q
$p \rightarrow q$	p implies q
$\forall p$	for all n, $p(n)$
$\exists p$	exists n, $p(n)$

The language of nonstandard PA is represented with nearly-equivalent data types `NPeanoExpr` (with one extra case: a symbol for a nonstandard number $\Bbbk$) and `NPeanoProp`. This possibly-suboptimal decision ended up requiring many functions and identities to convert between the two during proofs. The full list of cases are displayed above.

One of the most difficult aspects of this project was handling scopes and variables. I chose to use the method of DeBruijn Indices, where `PeanoExpr` v_i represents the variable of the i^{th} quantifier above (yes, one-indexed; more on this later). For example, the term $\forall\exists\,(v_1 = v_2)$: `PeanoProp` represents $\forall x.\exists y.y = x$ as stated in conventional notation. DeBruijn Indices have many advantages over named variables, especially in functional languages, but there are still many challenges reasoning about this representation. My implementation was inspired by [1].

2.2 Property Functions

Analyzing the language of PA requires functions to test or extract different properties of expressions and propositions, and if that makes your mouth water then this section is your proverbial jumbo bucket of wings.

No Zero Variables The predicate ✓ verifies that a `PeanoExpr` or `PeanoProp` doesn't contain any v_0 instances. Because I used one-indexed DeBruijn Indices, v_0 is reserved as a placeholder for performing substitutions and thus is not a valid expression in the language of PA. This implementation detail has its pros and cons. This predicate can also be extended to a list of `PeanoProp` to check them all at once. For completeness, the full specification is listed below.

$$\begin{aligned}
\checkmark 0 &\triangleq \texttt{True}\\
\checkmark Sx &\triangleq \checkmark x\\
\checkmark (x+y) &\triangleq (\checkmark x)\ \texttt{and}\ (\checkmark y)\\
\checkmark (x\times y) &\triangleq (\checkmark x)\ \texttt{and}\ (\checkmark y)\\
\checkmark v_i &\triangleq (i \neq 0)\\
\checkmark \Bbbk &\triangleq \texttt{True}\\
\checkmark \top &\triangleq \texttt{True}\\
\checkmark \bot &\triangleq \texttt{True}\\
\checkmark \neg p &\triangleq \checkmark p\\
\checkmark (x=y) &\triangleq (\checkmark x)\ \texttt{and}\ (\checkmark y)\\
\checkmark (p\vee q) &\triangleq (\checkmark p)\ \texttt{and}\ (\checkmark q)\\
\checkmark (p\wedge q) &\triangleq (\checkmark p)\ \texttt{and}\ (\checkmark q)\\
\checkmark (p\to q) &\triangleq (\checkmark p)\ \texttt{and}\ (\checkmark q)\\
\checkmark \forall p &\triangleq \checkmark p\\
\checkmark \exists p &\triangleq \checkmark p\\
\checkmark [] &\triangleq \texttt{True}\\
\checkmark (h :: \Gamma) &\triangleq (\checkmark h)\ \texttt{and}\ (\checkmark \Gamma)
\end{aligned}$$

References DeBruijn Index The predicate $\texttt{ref}_j$ tests if an index j is referenced anywhere, taking scope into account; the missing cases are obvious. You will not see this much.

$$\begin{aligned}
\texttt{ref}_j\,(v_i) &\triangleq (i=j)\\
\texttt{ref}_j\,(x+y) &\triangleq (\texttt{ref}_j x)\ \texttt{or}\ (\texttt{ref}_j y)\\
\texttt{ref}_j\,(\forall p) &\triangleq \texttt{ref}_{j+1}\,(p)\\
\texttt{ref}_j\,(\exists p) &\triangleq \texttt{ref}_{j+1}\,(p)
\end{aligned}$$

Minimum Scope An actually important property is the $[\![\cdot]\!]$ function, which equals the maximum index of any referenced free variable (or 0 if closed), thus the minimum size scope of free variables needed to contain it; the missing cases are just as obvious as before. Here, I use the "Pseudo-Predecessor" function `pred` with $\texttt{pred}\,(0) = 0$ and $\texttt{pred}\,(Sn) = n$.

$$\begin{aligned}
[\![v_i]\!] &\triangleq i\\
[\![x+y]\!] &\triangleq \max\left([\![x]\!], [\![y]\!]\right)\\
[\![\forall p]\!] &\triangleq \texttt{pred}\,([\![p]\!])\\
[\![\exists p]\!] &\triangleq \texttt{pred}\,([\![p]\!])
\end{aligned}$$

Valid Scope However, `pred` is very annoying when doing inductive proofs. It is often more convenient to use a combined predicate which tests $[\![\cdot]\!] \leq d$ for a particular d, since this is almost always what I really care about. This formulation is proven equivalent to separately calculating then testing (by Lemma 3.4), so there won't be a new notation.

$$\begin{aligned}
[\![v_i]\!] \leq d &\triangleq i \leq d\\
[\![x+y]\!] \leq d &\triangleq ([\![x]\!] \leq d)\ \texttt{and}\ ([\![y]\!] \leq d)\\
[\![\forall p]\!] \leq d &\triangleq [\![p]\!] \leq d+1\\
[\![\exists p]\!] \leq d &\triangleq [\![p]\!] \leq d+1
\end{aligned}$$

Numeral Embedding To convert from an i: `nat` to a `PeanoExpr` numeral representing the same natural number, I will use conventional iterated function notation $S^i 0$.

2.3 Transformation Functions

Reasoning about `PeanoExpr` and `PeanoProp` in context of the axioms of PA requires a variety of functions to transform them, and this section is the all-you-can-eat buffet. Missing cases will always be obvious.

Increment Free Variables The function $\uparrow^d$ recursively increments the DeBruijn Indices of all free variables to an expression e or proposition p within d quantifiers. This is used when I want to put something inside a quantifier without breaking variable references.

$$\begin{aligned}
\uparrow^d v_i &\triangleq \begin{cases} v_i & \text{if } i < d\\ v_{i+1} & \text{if } i \geq d \end{cases}\\
\uparrow^d (x+y) &\triangleq \left(\uparrow^d x\right) + \left(\uparrow^d y\right)\\
\uparrow^d \forall p &\triangleq \forall \uparrow^{d+1} p\\
\uparrow^d \exists p &\triangleq \exists \uparrow^{d+1} p
\end{aligned}$$

As an example use-case, I define syntactic sugar below for `PeanoProp` versions of the inequality operators, where $\uparrow^1$ is used to free-up the variable v_1 to be the distance from x to y.

$$\begin{aligned}
x \leq y &\triangleq \exists\left(\left(\uparrow^1 x\right) + v_1 = \left(\uparrow^1 y\right)\right)\\
x < y &\triangleq \exists\left(\left(\uparrow^1 x\right) + Sv_1 = \left(\uparrow^1 y\right)\right)
\end{aligned}$$

Decrement Free Variables Similarly, the function $\downarrow^d$ decrements those indices, except it replaces v_d with v_0 for the purpose of substitution.

$$\begin{aligned}
\downarrow^d v_i &\triangleq \begin{cases} v_i & \text{if } i < d\\ v_0 & \text{if } i = d\\ v_{i-1} & \text{if } i > d \end{cases}\\
\downarrow^d (x+y) &\triangleq \left(\downarrow^d x\right) + \left(\downarrow^d y\right)\\
\downarrow^d \forall p &\triangleq \forall \downarrow^{d+1} p\\
\downarrow^d \exists p &\triangleq \exists \downarrow^{d+1} p
\end{aligned}$$

It's clear that this is the left inverse, where $\downarrow^d\uparrow^d p = p$ as long as p contains no v_0, but this incredibly profound identity (Lemma 3.7) is rarely needed.

Placeholder Substitution Another important function is ☼_e, which replaces all occurrences of v_0 with a `PeanoExpr` e, which

has its free variables incremented as needed.

$$\begin{aligned}
\Sigma_e v_i &\triangleq \begin{cases} e & \text{if } i = 0 \\ v_i & \text{if } i > 0 \end{cases} \\
\Sigma_e (x + y) &\triangleq (\Sigma_e x) + (\Sigma_e y) \\
\Sigma_e \forall p &\triangleq \forall \Sigma_{(\uparrow^1 e)} p \\
\Sigma_e \exists p &\triangleq \exists \Sigma_{(\uparrow^1 e)} p
\end{aligned}$$

Beta Substitution The preceding functions allow us to define $a\,(e) \triangleq \Sigma_e \downarrow^1 a$. This allows us to "apply" a term with index 1 free to a given `PeanoExpr` e. For example, if p is $\forall\,(v_1 = v_2)$, which has index 1 free, then we have:

$$p\,(e) \triangleq \Sigma_e \downarrow^1 \forall\,(v_1 = v_2) \triangleq \forall\,\big(v_1 = (\uparrow^1 e)\big)$$

Put an S in Front of Every Reference to DeBruijn Index d On a related note, for representing induction, it was nice to have this function σ^d, which does what it says on the tin.

$$\begin{aligned}
\sigma^d v_i &\triangleq \begin{cases} S v_i & \text{if } i = d \\ v_i & \text{if } i \neq d \end{cases} \\
\sigma^d\,(x + y) &\triangleq \big(\sigma^d x\big) + \big(\sigma^d y\big) \\
\sigma^d \forall p &\triangleq \forall \sigma^{d+1} p \\
\sigma^d \exists p &\triangleq \exists \sigma^{d+1} p
\end{aligned}$$

Standard to Nonstandard Conversion Everything in standard PA is valid in nonstandard PA, and this conversion is traditionally noted with a superscript $*$, such as $\top^*$. However, the other direction needs something to replace any occurrences of the nonstandard symbol $\mathbb{k}$. For my purposes, I replace every $\mathbb{k}$ with the literal $S^k 0$ for a given k: `nat` using the conversion function $\natural^k$. Again, I show this is the left inverse: for all k, $\natural^k\,(a^*) = a$ (Lemma 3.29).

Standard to Coq Conversion For formal verification, I also need to be able to convert valid `PeanoExpr` and `PeanoProp` into equivalent `nat` and `Set`[1], respectively. Given $\mathtt{L} : \mathtt{list\ nat}$ of free variable values, this conversion is done with the $\mathtt{L}(\!|\cdot|\!)$ function, described below. Notably, if $\checkmark e$ and $[\![e]\!] \leq \mathtt{length\ L}$, then $\mathtt{L}(\!|e|\!)$ will have every occurrence of v_i replaced by the i^{th} element of L, with no index-out-of-bounds issues.

$$\begin{aligned}
\mathtt{L}(\!|0|\!) &\triangleq \mathtt{0} \\
\mathtt{L}(\!|Sx|\!) &\triangleq \mathtt{S}\,(\mathtt{L}(\!|x|\!)) \\
\mathtt{L}(\!|x + y|\!) &\triangleq \mathtt{L}(\!|x|\!) +_{\mathtt{nat}} \mathtt{L}(\!|y|\!) \\
\mathtt{L}(\!|x \times y|\!) &\triangleq \mathtt{L}(\!|x|\!) \times_{\mathtt{nat}} \mathtt{L}(\!|y|\!) \\
\mathtt{L}(\!|v_i|\!) &\triangleq \mathtt{nth_default}\,(\mathtt{0}, \mathtt{0} :: \mathtt{L}, i) \\
\mathtt{L}(\!|\top|\!) &\triangleq \mathtt{True} \\
\mathtt{L}(\!|\bot|\!) &\triangleq \mathtt{False} \\
\mathtt{L}(\!|\neg p|\!) &\triangleq \mathtt{L}(\!|p|\!) \to \mathtt{False} \\
\mathtt{L}(\!|x = y|\!) &\triangleq \mathtt{L}(\!|x|\!) = \mathtt{L}(\!|y|\!) \\
\mathtt{L}(\!|p \vee q|\!) &\triangleq \mathtt{L}(\!|p|\!) +_{\mathtt{Set}} \mathtt{L}(\!|q|\!) \\
\mathtt{L}(\!|p \wedge q|\!) &\triangleq \mathtt{L}(\!|p|\!) \times_{\mathtt{Set}} \mathtt{L}(\!|q|\!) \\
\mathtt{L}(\!|p \to q|\!) &\triangleq \mathtt{L}(\!|p|\!) \to \mathtt{L}(\!|q|\!) \\
\mathtt{L}(\!|\forall p|\!) &\triangleq \prod_{v:\mathtt{nat}} (v :: \mathtt{L})\,(\!|p|\!) \\
\mathtt{L}(\!|\exists p|\!) &\triangleq \sum_{v:\mathtt{nat}} (v :: \mathtt{L})\,(\!|p|\!) \\
\mathtt{L}(\!|[]|\!) &\triangleq \mathtt{True} \\
\mathtt{L}(\!|h :: \Gamma|\!) &\triangleq \mathtt{L}(\!|h|\!) \times_{\mathtt{Set}} \mathtt{L}(\!|\Gamma|\!)
\end{aligned}$$

[1] `Set` was chosen over `Prop` because in Coq, proofs of a `Prop` only exist at the type level. For example, if H : `exists v, P v` then one cannot extract `v` from H and print it to the user . . . what skulduggery is that?! Why even bother with *constructive* logic if it doesn't let you *construct*?

Standard to Nonstandard Conversion 2 I also introduce a new transformation function $\maltese^d$, which standardizes by recursively replacing $\mathbb{k}$ with a variable v_d, respecting scopes.

$$\begin{aligned}
\maltese^d \mathbb{k} &\triangleq v_d \\
\maltese^d \forall p &\triangleq \forall \maltese^{d+1} p \\
\maltese^d \exists p &\triangleq \exists \maltese^{d+1} p
\end{aligned}$$

List Insertion I also define these functions to insert an item into a list, since Coq does not include them.

$$\begin{aligned}
\mathtt{nth_insert}\,(\mathtt{L}, \mathtt{x}, \mathtt{n}) &\triangleq \mathtt{firstn}\,(\mathtt{L}, \mathtt{n})\ \mathtt{++}\ [\mathtt{x}]\ \mathtt{++skipn}\,(\mathtt{L}, \mathtt{n}) \\
\mathtt{push}\,(\mathtt{L}, \mathtt{x}) &\triangleq \mathtt{L++}\,[\mathtt{x}]
\end{aligned}$$

2.4 Axioms

Given the language of Peano Arithmetic, I can now formalize its axioms in the natural deduction style. I write the underlying parameterized data type $(\mathtt{N})\mathtt{PeanoTheorem}(d, \Gamma, p)$ as $\Gamma \vdash^d p$, meaning that p is provable from the ordered list of assumptions Γ in the context of d free variables. Parameterizing on d was a novel and very impactful decision that is discussed further in Section 7. The deduction rules are thus represented by the constructors for $\Gamma \vdash^d p$, listed in Table 1. The choice of axioms was influenced by [2, 4, 5, 7].

Proof Properties To analyze these axioms, given a nonstandard $\mathtt{thm} : \Gamma \vdash^d p$, it will be useful to define the "Maximum Nonstandard Axiom" function $\wr\cdot\int$ that picks out the largest i: `nat` of all applications of $\mathtt{Nbig}\,(i, \ldots)$ rules in `thm`, or 0 if none exist. I also define the "Is It Constructive?" predicate `constr` which tests for the use of `LEM` in `thm`. Implementations left to the reader, but make sure you don't burn yourself on the hot oil!

3. Lemmas À La Carte

If you aren't hungry enough for a full Theorem, browse the largest menu of bite-sized Lemmas this side of the Mississippi River! They are the core ingredients of the main results in Section 4.

3.1 The Cold Hard Truth

Lemma 3.1. $\Sigma_{v_0} a = a$.

Lemma 3.2. *If $\checkmark e$ then $\Sigma_x e = e$.*

Lemma 3.3. *If $\checkmark e$ then $\checkmark \Sigma_x e$.*

Lemma 3.4. *$[\![a]\!] \leq d$ ("Minimum Scope") if and only if $[\![a]\!] \leq d$ ("Valid Scope").*

Lemma 3.5. *If $[\![x]\!] \leq d$ and $[\![f]\!] \leq d$ then $[\![\Sigma_x f]\!] \leq d$.*

Lemma 3.6. *If $[\![x]\!] \leq d$ and $[\![p]\!] \leq d + 1$ then $[\![p\,(x)]\!] \leq d$.*

Lemma 3.7. $\downarrow^d \uparrow^d a = a$.

Lemma 3.8. *If $\checkmark a$ then $a = \Sigma_{(v_{d+1})} \downarrow^{d+1} \uparrow^{d+2} a$. In particular, we have $a = \big(\uparrow^2 a\big)\,(v_1)$.*

Lemma 3.9. *If $\checkmark a$ and $\sim \mathtt{ref}_{d+1} a$ then $a = \Sigma_x \downarrow^{d+1} \uparrow^{d+2} a$. In particular, if $\sim \mathtt{ref}_1 a$ then we have $a = \big(\uparrow^2 a\big)\,(x)$.*

Lemma 3.10. $\uparrow^d S^k 0 = S^k$.

3.2 Standard to Nonstandard Conversion Identities

Lemma 3.11. *$a = b$ if and only if $a^* = b^*$.*

Lemma 3.12. *$\checkmark a$ if and only if $\checkmark a^*$.*

Lemma 3.13. $[\![a]\!] = [\![a^*]\!]$.

Lemma 3.14. $\big(\uparrow^d a\big)^* = \uparrow^d a^*$.

$\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d \quad \checkmark h \quad \lVert h\rVert \le d}{h :: \Gamma \vdash^d h}$(HypoUse)	$\dfrac{\checkmark h \quad \lVert h\rVert \le d \quad \Gamma \vdash^d p}{h :: \Gamma \vdash^d p}$(HypoLift)	$\dfrac{\Gamma \vdash^d p}{\uparrow^1 \Gamma \vdash^{(d+1)} \uparrow^1 p}$(ScopeLift)
$\dfrac{\Gamma \vdash^d Sx = Sy}{\Gamma \vdash^d x = y}$(EqS)	$\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d \quad \checkmark x \quad \lVert x\rVert \le d}{\Gamma \vdash^d \neg(Sx = 0)}$(SneqZ)	$\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d \quad \checkmark x \quad \lVert x\rVert \le d}{\Gamma \vdash^d x = 0 \vee \exists(\uparrow^1 x = Sv_1)}$(SorZ)
$\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d \quad \checkmark x \quad \lVert x\rVert \le d}{\Gamma \vdash^d x = x}$(EqRefl)	$\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d \quad \checkmark x \quad \lVert x\rVert \le d}{\Gamma \vdash^d (x+0) = x}$(PlusZ)	$\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d \quad \checkmark x \quad \lVert x\rVert \le d}{\Gamma \vdash^d (x \times 0) = 0}$(TimesZ)
$\dfrac{\lVert p\rVert \le d \quad \Gamma \vdash^d x = y}{\Gamma \vdash^d (\dot\varpi_x p) \to (\dot\varpi_y p)}$(EqSubP)	$\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d \quad \checkmark x \quad \lVert x\rVert \le d \quad \checkmark y \quad \lVert y\rVert \le d}{\Gamma \vdash^d (x + Sy) = S(x+y)}$(PlusS)	$\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d \quad \checkmark x \quad \lVert x\rVert \le d \quad \checkmark y \quad \lVert y\rVert \le d}{\Gamma \vdash^d (x \times Sy) = x + (x \times y)}$(TimesS)
$\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d}{\Gamma \vdash^d \top}$(ATrue)	$\dfrac{p :: \Gamma \vdash^d q}{\Gamma \vdash^d p \to q}$(ImpIntro)	$\dfrac{\Gamma \vdash^d p \to \bot}{\Gamma \vdash^d \neg p}$(NotIntro)
$\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d \quad \checkmark p \quad \lVert p\rVert \le d}{\Gamma \vdash^d \bot \to p}$(AFalse)	$\dfrac{\Gamma \vdash^d p \to q \quad \Gamma \vdash^d p}{\Gamma \vdash^d q}$(ImpElim)	$\dfrac{\Gamma \vdash^d \neg p}{\Gamma \vdash^d p \to \bot}$(NotElim)
$\dfrac{\Gamma \vdash^d p \wedge q}{\Gamma \vdash^d p}$(AndLeft)	$\dfrac{\Gamma \vdash^d p \wedge q}{\Gamma \vdash^d q}$(AndRight)	$\dfrac{\Gamma \vdash^d p \quad \Gamma \vdash^d q}{\Gamma \vdash^d p \wedge q}$(AndIntro)
$\dfrac{\checkmark p \quad \lVert p\rVert \le d \quad \Gamma \vdash^d q}{\Gamma \vdash^d p \vee q}$(OrLeft)	$\dfrac{\checkmark q \quad \lVert q\rVert \le d \quad \Gamma \vdash^d p}{\Gamma \vdash^d p \vee q}$(OrRight)	$\dfrac{\Gamma \vdash^d p \vee q \quad \Gamma \vdash^d p \to r \quad \Gamma \vdash^d q \to r}{\Gamma \vdash^d r}$(OrElim)
$\dfrac{\Gamma \vdash^d p(0) \quad \Gamma \vdash^d \forall\left(p \to \sigma^1 p\right)}{\Gamma \vdash^d \forall p}$(Induct)	$\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d \quad \uparrow^1 \Gamma \vdash^{(d+1)} p}{\Gamma \vdash^d \forall p}$(ForGen)	$\dfrac{\checkmark x \quad \lVert x\rVert \le d \quad \Gamma \vdash^d \forall p}{\Gamma \vdash^d p(x)}$(ForApp)
$\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d \quad \checkmark p \quad \lVert p\rVert \le d}{\Gamma \vdash^d p \vee \neg p}$(LEM)	$\dfrac{\checkmark q \quad \Gamma \vdash^d \exists p \quad \uparrow^1 \Gamma \vdash^{(d+1)} p \to \uparrow^1 q}{\Gamma \vdash^d q}$(ExElim)	$\dfrac{\checkmark x \quad \lVert x\rVert \le d \quad \Gamma \vdash^d p(x)}{\Gamma \vdash^d \exists p}$(Exhibit)
for all $i : nat$, $\dfrac{\checkmark\Gamma \quad \lVert\Gamma\rVert \le d}{\Gamma \vdash^d S^i 0 < \mathbb{k}}$ (Nbig, only present for NPeanoTheorem)		

Table 1. The constructors of PeanoTheorem (d, Γ, p), written as $\Gamma \vdash^d p$, which represent PA's deduction rules. The constructors of NPeanoTheorem (d, Γ, p) are the same (except the names are prepended with an N) with one additional rule Nbig.

Lemma 3.15. $\left(\downarrow^d a\right)^* = \downarrow^d a^*$.

Lemma 3.16. $(\dot\varpi_e a)^* = \dot\varpi_{e^*} a^*$.

Lemma 3.17. $(a\,(e))^* = a^*\,(e^*)$.

Lemma 3.18. $\left(\sigma^d a\right)^* = \sigma^d a^*$.

3.3 Nonstandard to Standard Conversion Identities

Lemma 3.19. $\lVert a\rVert = \lVert \looparrowright^k a\rVert$.

Lemma 3.20. *$\checkmark a$ if and only if $\checkmark \looparrowright^k a$.*

Lemma 3.21. $\looparrowright^k \uparrow^d a = \uparrow^d \looparrowright^k a$.

Lemma 3.22. $\looparrowright^k \left(\uparrow^1\right)^d a = \left(\uparrow^1\right)^d \looparrowright^k a$.

Lemma 3.23. *If $d \le k$ then* $\looparrowright^k \downarrow^d a = \downarrow^d \looparrowright^k a$.

Lemma 3.24. $\looparrowright^k \dot\varpi_v a = \dot\varpi_{\left(\looparrowright^k v\right)} \looparrowright^k a$.

Lemma 3.25. $\looparrowright^k p\,(v) = \left(\looparrowright^k p\right)\left(\looparrowright^k v\right)$.

Lemma 3.26. $\looparrowright^k \sigma^d a = \sigma^d \looparrowright^k a$.

Lemma 3.27. $\looparrowright^k (a < b) = \left(\looparrowright^k a\right) < \left(\looparrowright^k b\right)$.

Lemma 3.28. $\looparrowright^k \left(S^i 0\right) = S^i 0$.

Lemma 3.29. $\looparrowright^k (a^*) = a$.

3.4 Nonstandard to Standard Conversion 2 Identities

Lemma 3.30. *If $d < k$ and $\lVert a\rVert \le d$ then $\lVert \curlywedge^k a\rVert \le k$.*

Lemma 3.31. *$\checkmark a$ if and only if $\checkmark \curlywedge^{k+1} a$.*

Lemma 3.32. *If $d \le k$ then* $\curlywedge^{k+1} \uparrow^d a = \uparrow^d \curlywedge^k a$.

Lemma 3.33. *If* $d \leq k$ *then* $\mathcal{N}^{k+1}\left(\uparrow^1\right)^d a = \left(\uparrow^1\right)^d \mathcal{N}^k a$.

Lemma 3.34. *If* $d \leq k$ *then* $\mathcal{N}^k \downarrow^d a = \downarrow^d \mathcal{N}^{k+1} a$.

Lemma 3.35. $\mathcal{N}^{k+1} \divideontimes_v a = \divideontimes_{\left(\mathcal{N}^{k+1} v\right)} \mathcal{N}^{k+1} a$.

Lemma 3.36. $\mathcal{N}^{k+1} p\,(v) = \left(\mathcal{N}^{k+2} p\right)\left(\mathcal{N}^{k+1} v\right)$.

Lemma 3.37. *If* $k \neq d$ *then* $\mathcal{N}^k \sigma^d a = \sigma^d \mathcal{N}^k a$.

Lemma 3.38. $\mathcal{N}^{d+1}\,(a < b) = \left(\mathcal{N}^{d+1} a\right) < \left(\mathcal{N}^{d+1} b\right)$.

Lemma 3.39. $\not{z}^k\left(S^i 0\right) = S^i 0$.

3.5 List Insert Properties

Lemma 3.40. *If* $\mathtt{i} < \mathtt{n}$ *and* $\mathtt{i} < \mathtt{length}\,(\mathtt{L})$ *then*

$$\mathtt{nth\_error}\,(\mathtt{nth\_insert}\,(\mathtt{L}, \mathtt{x}, \mathtt{n})\,, \mathtt{i}) = \mathtt{nth\_error}\,(\mathtt{L}, \mathtt{i})$$

Lemma 3.41. *If* $\mathtt{n} \leq \mathtt{length}\,(\mathtt{L})$ *then*

$$\mathtt{nth\_error}\,(\mathtt{nth\_insert}\,(\mathtt{L}, \mathtt{x}, \mathtt{n})\,, \mathtt{n}) = \mathtt{Some}\,(\mathtt{x})$$

Lemma 3.42. *If* $\mathtt{n} \leq \mathtt{i}$ *then*

$$\mathtt{nth\_error}\,(\mathtt{nth\_insert}\,(\mathtt{L}, \mathtt{x}, \mathtt{n})\,, \mathtt{i}+1) = \mathtt{nth\_error}\,(\mathtt{L}, \mathtt{i})$$

Lemma 3.43. *If* $\mathtt{length}\ \mathtt{l} \leq \mathtt{n}$ *then*

$$\mathtt{push}\,(\mathtt{l}, \mathtt{x}) = \mathtt{nth\_insert}\,(\mathtt{l}, \mathtt{x}, \mathtt{n})$$

3.6 Theorem Syntactic Correctness

Lemma 3.44. *Given* $\mathtt{thm} : \Gamma \vdash^d p$, *we have* $\llbracket \Gamma \rrbracket \leq d$ *and* $\llbracket p \rrbracket \leq d$.

Lemma 3.45. *Given* $\mathtt{thm} : \Gamma \vdash^d p$, *we have* $\checkmark \Gamma$ *and* $\checkmark p$.

3.7 Conversion Fold Relations

Lemma 3.46. $\mathtt{L}(\!|\mathtt{foldr}\,(\Gamma, \to, p)|\!)$ *if and only if* $\mathtt{L}(\!|\Gamma|\!) \to \mathtt{L}(\!|p|\!)$.

Lemma 3.47. $\llbracket \mathtt{foldr}\,(\Gamma, \to, p) \rrbracket \leq d$ *if and only if* $\llbracket \Gamma \rrbracket \leq d$ *and* $\llbracket p \rrbracket \leq d$.

Lemma 3.48. $\mathtt{foldr}\left(\uparrow^d \Gamma, \to, \uparrow^d p\right) = \uparrow^d \mathtt{foldr}\,(\Gamma, \to, p)$.

Lemma 3.49. *If* $d \leq \mathtt{length}\,(\mathtt{L})$ *and* $\checkmark \Gamma$ *and* $\llbracket \Gamma \rrbracket \leq d$ *and* $\mathtt{v} : \mathtt{nat}$, *then* $\mathtt{L}(\!|\Gamma|\!) \to (\mathtt{v} :: \mathtt{L})(\!|\uparrow^1 \Gamma|\!)$.

3.8 Coq Conversion Lemmas

Lemma 3.50. *If* $\checkmark e$ *and* $\llbracket e \rrbracket \leq \mathtt{length}\,(\mathtt{L})$ *and* $d \leq \mathtt{length}\,(\mathtt{L})$, *then* $\mathtt{L}(\!|e|\!) = \mathtt{nth\_insert}\,(\mathtt{L}, \mathtt{v}, \mathtt{d})(\!|\uparrow^{d+1} e|\!)$.

Lemma 3.51. *If* $\checkmark p$ *and* $\llbracket p \rrbracket \leq \mathtt{length}\,(\mathtt{L})$ *and* $d \leq \mathtt{length}\,(\mathtt{L})$, *then* $\mathtt{L}(\!|p|\!)$ *if and only if* $\mathtt{nth\_insert}\,(\mathtt{L}, \mathtt{v}, \mathtt{d})(\!|\uparrow^{d+1} p|\!)$.

Lemma 3.52. *If* $\mathtt{L}(\!|x|\!) = \mathtt{L}(\!|y|\!)$ *then* $\mathtt{L}(\!|\divideontimes_x e|\!) = \mathtt{L}(\!|\divideontimes_y e|\!)$.

Lemma 3.53. *If* $\mathtt{L}(\!|x|\!) = \mathtt{L}(\!|y|\!)$ *then* $\mathtt{L}(\!|\divideontimes_x p|\!)$ *if and only if* $\mathtt{L}(\!|\divideontimes_y p|\!)$.

Lemma 3.54. $\mathtt{L}(\!|\divideontimes_x \downarrow^{n+1} e|\!) = \mathtt{nth\_insert}\,(\mathtt{L}, \mathtt{L}(\!|x|\!), n)\,(\!|e|\!)$ *if* $\checkmark e$ *and* $n \leq \mathtt{length}\,(\mathtt{L})$.

Lemma 3.55. *If* $\checkmark p$ *and* $n \leq \mathtt{length}\,(\mathtt{L})$, *for* $x^n = \left(\uparrow^1\right)^n x$ *we have* $\mathtt{L}(\!|\divideontimes_{x^n} \downarrow^{n+1} p|\!)$ *if and only if* $\mathtt{nth\_insert}\,(\mathtt{L}, \mathtt{L}(\!|x^n|\!), n)\,(\!|p|\!)$.

Lemma 3.56. *If* $\checkmark p$ *then* $\mathtt{L}(\!|p\,(x)|\!)$ *if and only if* $(\mathtt{L}(\!|x|\!) :: \mathtt{L})\,(\!|p|\!)$.

Lemma 3.57. *If* $\checkmark p$ *then* $\mathtt{nth\_insert}\,(\mathtt{L}, \mathtt{v}, n)\,(\!|\sigma^{n+1} p|\!)$ *if and only if* $\mathtt{nth\_insert}\,(\mathtt{L}, \mathtt{v}+1, n)\,(\!|p|\!)$. *In particular, for* $n = 0$, *we get that* $(\mathtt{v} :: \mathtt{L})(\!|\sigma^1 p|\!)$ *if and only if* $((\mathtt{v}+1) :: \mathtt{L})(\!|p|\!)$.

3.9 Formalized Arithmetic Derivations

Lemma 3.58. *If* $\llbracket f \rrbracket \leq d$ *and* $\Gamma \vdash^d x = y$ *then*

$$\Gamma \vdash^d \divideontimes_x f = \divideontimes_y f$$

Proof. An explicit proof is given in Figure 3.1. □

Lemma 3.59. *If* $\Gamma \vdash^d x = y$ *then* $\Gamma \vdash^d y = x$.

Proof. An explicit proof is given in Figure 3.2. □

Lemma 3.60. *If* $\Gamma \vdash^d x = y$ *and* $\Gamma \vdash^d y = z$ *then* $\Gamma \vdash^d x = z$.

Proof. An explicit proof is given in Figure 3.3. □

Lemma 3.61. *If* $\Gamma \vdash^d x < y$ *then* $\Gamma \vdash^d x < Sy$.

Proof. An explicit proof is given in Figure 3.4. □

Lemma 3.62. *If* $i < k$, *then* $\Gamma \vdash^d S^i 0 < S^k 0$ *as long as* $\checkmark \Gamma$ *and* $\llbracket \Gamma \rrbracket \leq d$.

Proof. I perform induction on k. Obviously $i < 0$ is a contradiction, so given $i < k+1$ and hypotheses $\mathtt{thm}_j : \Gamma \vdash^d S^j 0 < S^k 0$ for all $j < k$, we want to show $\Gamma \vdash^d S^i 0 < S^{k+1} 0$. If $i < k$, then we can just apply Lemma 3.61 to $\mathtt{thm}_i : \Gamma \vdash^d S^i 0 < S^k 0$ and we're done! However, if $i = k$, then I provide an explicit proof of $\Gamma \vdash^d S^k 0 < S^{k+1} 0$ in Figure 3.5. □

3.10 Formalized Logical Derivations

Lemma 3.63. *If* $\Gamma \vdash^d \exists p$ *and* $\uparrow^1 \Gamma \vdash^{d+1} p \to \uparrow^1 \exists q$ *then* $\Gamma \vdash^d \exists q$.

Proof. Did you know that the existential quantifier was a monad? This "flat map"-like lemma follows directly from `ExElim`. □

Lemma 3.64. *If* $\Gamma \vdash^d p \to q$ *and* $\Gamma \vdash^d q \to r$, *then* $\Gamma \vdash^d p \to r$

Proof. An explicit proof is given in Figure 3.6. □

Lemma 3.65. *If* $\Gamma \vdash^d \exists p$ *and* $\uparrow^1 \Gamma \vdash^{d+1} p \to q$ *then* $\Gamma \vdash^d \exists q$.

Proof. An explicit proof is given in Figure 3.7. □

Lemma 3.66. *If* $\Gamma \vdash^d p \to q$ *then* $p :: \Gamma \vdash^d q$.

Proof. An explicit proof is given in Figure 3.8. □

Lemma 3.67. *If* $\Gamma \vdash^d p \to (q \to r)$ *then* $\Gamma \vdash^d q \to (p \to r)$. *Corrolarolly, if* $p :: q :: \Gamma \vdash^d r$ *then* $q :: p :: \Gamma \vdash^d r$.

Proof. An explicit proof is given in Figure 3.9. □

Lemma 3.68. *If* $L \mathbin{+\!\!+} [h] \mathbin{+\!\!+} \Gamma, n \vdash^d p$ *then* $h :: L \mathbin{+\!\!+} \Gamma \vdash^d r$.

Proof. I perform induction on the length of L. When $L = []$ it is trivially true, and to show length ℓ implies length $\ell + 1$, an explicit proof is given in Figure 3.10. □

Lemma 3.69. *If* $h :: L \mathbin{+\!\!+} \Gamma \vdash^d r$ *then* $L \mathbin{+\!\!+} [h] \mathbin{+\!\!+} \Gamma, n \vdash^d p$.

Proof. Suspiciously similar to Lemma 3.68, induction on the length of L is trivial when $L = []$ and an explicit proof of the inductive step is given in Figure 3.11. □

Lemma 3.70. *If* $\mathtt{nth\_insert}\,(\Gamma, p, n) \vdash^d r$ *and* $\Gamma \vdash^d q \to p$ *then* $\mathtt{nth\_insert}\,(\Gamma, q, n) \vdash^d r$.

Proof. An explicit proof is given in Figure 3.12. □

$$\dfrac{\dfrac{\checkmark\Gamma \quad \llbracket\Gamma\rrbracket \leq d \quad \checkmark \divideontimes_x f \quad \llbracket \divideontimes_x f \rrbracket \leq d}{\Gamma \vdash^d \divideontimes_x f = \divideontimes_x f}(\texttt{EqRefl}) \quad \dfrac{\dfrac{\llbracket \divideontimes_x f = \divideontimes_{v_0} f \rrbracket \leq d \quad \Gamma \vdash^d x = y}{\Gamma \vdash^d \divideontimes_x (\divideontimes_x f = \divideontimes_{v_0} f) \to \divideontimes_y (\divideontimes_x f = \divideontimes_{v_0} f)}(\texttt{EqSubP})}{\Gamma \vdash^d \divideontimes_x f = \divideontimes_x f \to \divideontimes_x f = \divideontimes_y f}(\text{simplify})}{\Gamma \vdash^d \divideontimes_x f = \divideontimes_y f}(\texttt{ImpElim})$$

Figure 3.1. Explicit Proof that Equal Substitutions are Equal, Lemma 3.58.

$$\dfrac{\dfrac{\checkmark\Gamma \quad \llbracket\Gamma\rrbracket \leq d \quad \checkmark x \quad \llbracket x \rrbracket \leq d}{\Gamma \vdash^d x = x}(\texttt{EqRefl}) \quad \dfrac{\dfrac{\llbracket v_0 = x \rrbracket \leq d \quad \Gamma \vdash^d x = y}{\Gamma \vdash^d \divideontimes_x (v_0 = x) \to \divideontimes_y (v_0 = x)}(\texttt{EqSubP})}{\Gamma \vdash^d x = x \to y = x}(\text{simplify})}{\Gamma \vdash^d y = x}(\texttt{ImpElim})$$

Figure 3.2. Explicit Proof of the Symmetry of Equality, Lemma 3.59.

$$\dfrac{\Gamma \vdash^d y = z \quad \dfrac{\dfrac{\llbracket v_0 = z \rrbracket \leq d \quad \dfrac{\Gamma \vdash^d x = y}{\Gamma \vdash^d y = x}(3.59)}{\Gamma \vdash^d \divideontimes_y (v_0 = z) \to \divideontimes_x (v_0 = z)}(\texttt{EqSubP})}{\Gamma \vdash^d y = z \to x = z}(\text{simplify})}{\Gamma \vdash^d x = z}(\texttt{ImpElim})$$

Figure 3.3. Explicit Proof of the Transitivity of Equality, Lemma 3.60.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x' + Sv_1 = y' \triangleq x' + Sv_1 = y'}{x' + Sv_1 = y' \triangleq (\uparrow^2 x')\,(Sv_1) + Sv_1 = (\uparrow^2 y')\,(Sv_1)}(3.9)}{x' + Sv_1 = y' \triangleq (\uparrow^2 (x' + v_1 = y'))\,(Sv_1)}}{(x' + Sv_1 = y') ::\uparrow^1 \Gamma \vdash^{d+1} (\uparrow^2 (x' + v_1 = y'))\,(Sv_1)}(\texttt{HypoUse})}{(x' + Sv_1 = y') ::\uparrow^1 \Gamma \vdash^{d+1} \exists \uparrow^2 (x' + v_1 = y')}(\texttt{Exhibit}\,(Sv_1))}{(x' + Sv_1 = y') ::\uparrow^1 \Gamma \vdash^{d+1}\uparrow^1 \exists\,(x' + v_1 = y')}}{\uparrow^1 \Gamma \vdash^{d+1} x' + Sv_1 = y' \to\uparrow^1 \exists\,(x' + v_1 = y')}(\texttt{ImpIntro})$$

$$\dfrac{\dfrac{\Gamma \vdash^d x < y}{\Gamma \vdash^d \exists\,(x' + Sv_1 = y')}(\text{def of } <, x', y') \quad \dfrac{\vdots}{\uparrow^1 \Gamma \vdash^{d+1} x' + Sv_1 = y' \to\uparrow^1 \exists\,(x' + v_1 = y')}(\text{above})}{\Gamma \vdash^d \exists\,(x' + v_1 = y')}(3.63)$$

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash^d \exists\,(x' + v_1 = y')}(\text{above}) \quad \dfrac{\dfrac{\dfrac{\text{done!}}{(x' + v_1 = y') ::\uparrow^1 \Gamma \vdash^{d+1} x' + v_1 = y'}(\texttt{HypoUse})}{(x' + v_1 = y') ::\uparrow^1 \Gamma \vdash^{d+1} \divideontimes_{(x' + v_1)} Sv_0 = \divideontimes_{(y')} Sv_0}(3.58)}{(x' + v_1 = y') ::\uparrow^1 \Gamma \vdash^{d+1} S\,(x' + v_1) = Sy'}}{\uparrow^1 \Gamma \vdash^{d+1} x' + v_1 = y' \to S\,(x' + v_1) = Sy'}(\texttt{ImpIntro})}{\Gamma \vdash^d \exists\,(S\,(x' + v_1) = Sy')}(3.65)$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash^d \exists\,(S\,(x' + v_1) = Sy')}(\text{above}) \quad \dfrac{\dfrac{\dfrac{\text{done!}}{\Gamma \vdash^d x' + Sv_1 = S\,(x' + v_1)}(\texttt{PlusS})}{\Gamma \vdash^d S\,(x' + v_1) = x' + Sv_1}(3.59)}{\Gamma \vdash^d \divideontimes_{S(x' + v_1)}\,(v_0 = Sy') \to \divideontimes_{(x' + Sv_1)}\,(v_0 = Sy')}(\texttt{EqSubP})}{\uparrow^1 \Gamma \vdash^{d+1} S\,(x' + v_1) = Sy' \to x' + Sv_1 = Sy'}}{\Gamma \vdash^d \exists\,(x' + Sv_1 = Sy')\text{, where } x' \triangleq\uparrow^1 x \text{ and } y' \triangleq\uparrow^1 y}(3.65)}{\Gamma \vdash^d x < Sy}(\text{def of } <)$$

Figure 3.4. Explicit Proof of whatever $x < y \to x < Sy$ is called, Lemma 3.60.

$$\dfrac{\dfrac{\dfrac{\dfrac{\checkmark\Gamma \quad \llbracket\Gamma\rrbracket \le d}{\Gamma \vdash^d S^k 0 + S0 = S\,(S^k 0 + 0)}\,(\texttt{PlusS}) \quad \dfrac{\dfrac{\checkmark\Gamma \quad \llbracket\Gamma\rrbracket \le d}{\Gamma \vdash^d S^k 0 + 0 = S^k 0}\,(\texttt{PlusZ})}{\Gamma \vdash^d \divideontimes_{(S^k 0+0)}\,(Sv_0) = \divideontimes_{(S^k 0)}\,(Sv_0)}\,(3.58)}{\Gamma \vdash^d S\,(S^k 0 + 0) = S^{k+1} 0}\,(\text{simplify})}{\Gamma \vdash^d S^k 0 + S0 = S^{k+1} 0}\,(3.60)}{\Gamma \vdash^d \exists\,(S^k 0 + Sv_1 = S^{k+1} 0)}\,(\texttt{Exhibit}\,(0))}{\Gamma \vdash^d S^k 0 < S^{k+1} 0}\,(\text{def of } <)$$

Figure 3.5. Explicit Proof that Successor Returns a Bigger Number, Lemma 3.62

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{done!}}{p :: \Gamma \vdash^d p}\,(\texttt{HypoUse}) \quad \dfrac{\Gamma \vdash^d p \to q}{p :: \Gamma \vdash^d p \to q}\,(\texttt{HypoLift})}{p :: \Gamma \vdash^d q}\,(\texttt{ImpElim}) \quad \dfrac{\Gamma \vdash^d q \to r}{p :: \Gamma \vdash^d q \to r}\,(\texttt{HypoLift})}{p :: \Gamma \vdash^d r}\,(\texttt{ImpElim})}{\Gamma \vdash^d p \to r}\,(\texttt{ImpIntro})$$

Figure 3.6. Explicit Proof of Transitivity of Implication, Lemma 3.64.

$$\dfrac{\Gamma \vdash^d \exists p \quad \dfrac{\uparrow^1 \Gamma \vdash^{d+1} p \to q \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{done!}}{q ::\uparrow^1 \Gamma \vdash^{d+1} q}\,(\texttt{HypoUse})}{q ::\uparrow^1 \Gamma \vdash^{d+1} (\uparrow^2 q)\,(v_1)}\,(3.8)}{q ::\uparrow^1 \Gamma \vdash^{d+1} \exists \uparrow^2 q}\,(\texttt{Exhibit}\,(v_1))}{q ::\uparrow^1 \Gamma \vdash^{d+1} \uparrow^1 \exists q}\,(\text{def of } \uparrow)}{\uparrow^1 \Gamma \vdash^{d+1} q \to \uparrow^1 \exists q}\,(\texttt{ImpIntro})}{\uparrow^1 \Gamma \vdash^{d+1} p \to \uparrow^1 \exists q}\,(3.64)}{\Gamma \vdash^d \exists q}\,(3.63)$$

Figure 3.7. Explicit Proof of Implication under Exists, Lemma 3.65.

$$\dfrac{\dfrac{\Gamma \vdash^d p \to q}{p :: \Gamma \vdash^d p \to q}\,(\texttt{HypoLift}) \quad \dfrac{\text{done!}}{p :: \Gamma \vdash^d p}\,(\texttt{HypoUse})}{p :: \Gamma \vdash^d q}\,(\texttt{ImpElim})$$

Figure 3.8. Explicit Proof of Unintroducing Implication, Lemma 3.66.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash^d p \to (q \to r)}{q :: \Gamma \vdash^d p \to (q \to r)}\,(\texttt{HypoLift})}{p :: q :: \Gamma \vdash^d q \to r}\,(3.66) \quad \dfrac{\dfrac{\text{done!}}{q :: \Gamma \vdash^d q}\,(\texttt{HypoUse})}{p :: q :: \Gamma \vdash^d q}\,(\texttt{HypoLift})}{q :: p :: \Gamma \vdash^d r}\,(\texttt{ImpElim})}{\Gamma \vdash^d q \to (p \to r)}\,(\texttt{ImpIntro},\ \text{twice})$$

Figure 3.9. Explicit Proof of Implication Commutation, Lemma 3.67.

$$\dfrac{\dfrac{\dfrac{(q :: L) +\!\!+ [h] +\!\!+ \Gamma \vdash^d p}{L +\!\!+ [h] +\!\!+ \Gamma \vdash^d q \to p}\,(\texttt{ImpIntro})}{h :: L +\!\!+ \Gamma \vdash^d q \to p}\,(\text{inductive hypothesis})}{q :: h :: L +\!\!+ \Gamma \vdash^d p}\,(3.66)}{h :: (q :: L) +\!\!+ \Gamma \vdash^d p}\,(3.67)$$

Figure 3.10. Explicit Proof of inductive step bringing h to the front, Lemma 3.68.

$$\dfrac{\dfrac{\dfrac{h :: (q :: L) +\!\!+ \Gamma \vdash^d p}{q :: h :: L +\!\!+ \Gamma \vdash^d p}\,(3.67)}{h :: L +\!\!+ \Gamma \vdash^d q \to p}\,(\texttt{ImpIntro})}{L +\!\!+ [h] +\!\!+ \Gamma \vdash^d q \to p}\,(\text{inductive hypothesis})}{(q :: L) +\!\!+ [h] +\!\!+ \Gamma \vdash^d p}\,(3.66)$$

Figure 3.11. Explicit Proof of inductive step removing h from the front, Lemma 3.68.

$$\dfrac{\dfrac{\Gamma \vdash^d q \to p \qquad \dfrac{\dfrac{\texttt{nth\_insert}\,(\Gamma, p, n) \vdash^d r}{p :: \Gamma \vdash^d r}\,(3.68)}{\Gamma \vdash^d p \to r}\,(\texttt{ImpIntro})}{\Gamma \vdash^d q \to r}\,(3.64)}{q :: \Gamma \vdash^d r}\,(\texttt{ImpIntroRev})$$
$$\dfrac{}{\texttt{nth\_insert}\,(\Gamma, q, n) \vdash^d r}\,(3.69)$$

Figure 3.12. Explicit Proof for rewriting a hypothesis, Lemma 3.70

3.11 Formalized Arithmetic Derivations V: The Empire Strikes Back

Lemma 3.71. *If* $\Gamma \vdash^d x = y$ *then* $\Gamma \vdash^d Sx = Sy$.

Proof. This follows directly from Lemma 3.58. □

Lemma 3.72. *If* $\Gamma \vdash^d S^x 0 + S^y 0 = S^{x+y} 0$, *assuming* $\checkmark\Gamma$ *and* $\lceil\lceil\Gamma\rceil\rceil \leq d$.

Proof. By induction on y, the base case is `PlusZ`, and the inductive step uses `PlusS` and Lemmas 3.60 and 3.71. □

Lemma 3.73. $\Gamma \vdash^d \forall\,(0 + v_1 = v_1 + 0)$, *as long as* $\checkmark\Gamma$ *and* $\lceil\lceil\Gamma\rceil\rceil \leq d$.

Proof. The commutativity of adding zero is proven with `Induct`. The base case is solved by `EqRefl`, and the inductive step is solved with `ForGen` and a gravy boat filled with Lemmas 3.60 and 3.59, `EqSubE` and `EqSubP`, `HypoUse`, `ImpIntro` and `ImpElim`, and most importantly `PlusZ` and `PlusS`. □

Lemma 3.74. $\Gamma \vdash^d \forall\forall\,(v_2 + v_1 = v_1 + v_1)$, *given* $\checkmark\Gamma$ *and* $\lceil\lceil\Gamma\rceil\rceil \leq d$.

Proof. The commutativity of addition is proven with `Induct`. The base case is Lemma 3.73, and the inductive step is solved with `ForGen` then another `Induct` with a dash of secret spices: `ForApp` and `ForGen`, Lemmas 3.60, 3.59, 3.73 and 3.71, `HypoLift` and `HypoUse`, `ImpIntro`, and `PlusS`. □

Lemma 3.75. $\Gamma \vdash^d Sx + y = S\,(x + y)$, *if all* $\checkmark$ *and* $\lceil\lceil\cdot\rceil\rceil \leq d$.

Proof. I show this reversed version of `PlusS` with Lemmas 3.60, 3.59, 3.71, and 3.74 (requiring `ForApp`) and `PlusS` itself. □

Lemma 3.76. *Given* $\checkmark\Gamma$ *and* $\lceil\lceil\Gamma\rceil\rceil \leq d$, *if* $i + 1 \leq d$, *then* $\Gamma \vdash^d S^x 0 + (S^y 0 + v_{i+1}) = (S^x 0 + S^y 0) + v_{i+1}$.

Proof. This limited associativity of addition is proven by induction on y. For $\Gamma \vdash^d S^x 0 + (0 + v_{i+1}) = (S^x 0 + 0) + v_{i+1}$, I mixed Lemma 3.73 and `PlusZ` in a pot with paprika and olive oil then bake at 425° for 30 minutes. While that's going, to get $\Gamma \vdash^d S^x 0 + \left(S^{y+1} 0 + v_{i+1}\right) = \left(S^x 0 + S^{y+1} 0\right) + v_{i+1}$ from the inductive hypothesis, I applied my famous dessert sauce: 1 liter of blended Lemma 3.75 with cinnamon and white chocolate chips. □

Lemma 3.77. *Given* $\checkmark\Gamma$ *and* $\lceil\lceil\Gamma\rceil\rceil \leq d + 1$, *we have:*

$$\Gamma \vdash^{d+1} \left(S^{\max(x,y)} 0 < v_{d+1}\right) \to (S^x 0 < v_{d+1})$$

Proof. This one has a long proof, so I'll cut to the chase. The core idea is to extract $S\alpha$, the distance from $S^{\max(x,y)} 0$ to v_{d+1}, then exhibit $S\left(S^{\max(x,y)-x} 0 + \alpha\right)$ as the distance from $S^x 0$ to v_{d+1}, then push the symbols around to prove the inequality. The critical steps are the uses of Lemmas 3.76 and 3.72. □

4. Signature Results

4.1 30 pc. InducTenders Box

I show that given any standard $\texttt{thm} : \Gamma \vdash^d p$, one can construct the corresponding `Set` in Coq.[2] However, there is one caveat: because PA includes the Law of Excluded Middle rule `LEM` but Coq uses constructive logic, one does require that `thm` is constructive to avoid assuming some classical logic.

Definition 4.1. *PALEM* is true if the Law of Excluded Middle is true for all `Set` that are "PA-like," meaning for all q : `PeanoProp` and `L` where $\checkmark q$ and $\lceil\lceil q\rceil\rceil \leq \texttt{length}\,(\texttt{L})$, we have $\texttt{L}(\!|q \vee \neg q|\!)$.

Theorem 4.2. *Given a standard* $\texttt{thm} : \Gamma \vdash^d p$ *and an* $\texttt{L} : \texttt{list nat}$ *such that* $d \leq \texttt{length}\,(\texttt{L})$, *and either* `constr (thm)` *or PALEM, we can prove* $\texttt{L}(\!|\Gamma|\!) \to \texttt{L}(\!|p|\!)$ *in Coq.*

Proof. By Lemma 3.46, I use $\texttt{L}(\!|\Gamma|\!) \to \texttt{L}(\!|p|\!)$ and $\texttt{L}(\!|\texttt{foldr}\,(\Gamma, \to, p)|\!)$ interchangeably. We also have that $\lceil\lceil\texttt{foldr}\,(\Gamma, \to, p)\rceil\rceil \leq d$ by Lemma 3.44. We perform induction on the proof `thm`, for each standard case in Table 1. For brevity, I will omit referencing $\texttt{L}(\!|\Gamma|\!)$ when it merely needs to be copied from a hypothesis, and omit proving $\lceil\lceil\texttt{foldr}\,(\Gamma, \to, p)\rceil\rceil \leq d \leq \texttt{length}\,(\texttt{L})$ and $\checkmark$ requirements, although they form a significant amount of the Coq code.

1. `HypoUse`: We must show $\texttt{L}(\!|h :: \Gamma|\!) \to \texttt{L}(\!|h|\!)$, which by definition is $\texttt{L}(\!|h|\!) \times \texttt{L}(\!|\Gamma|\!) \to \texttt{L}(\!|h|\!)$, which is clearly true since we can ignore the right-hand argument.
2. `HypoLift`: Given inductive hypothesis $\texttt{L}(\!|\Gamma|\!) \to \texttt{L}(\!|p|\!)$, it easily follows that $\texttt{L}(\!|h :: \Gamma|\!) \to \texttt{L}(\!|p|\!)$ since it equals $\texttt{L}(\!|h|\!) \times \texttt{L}(\!|\Gamma|\!) \to \texttt{L}(\!|p|\!)$ and we can ignore the left-hand argument and just use the inductive hypothesis.
3. `ScopeLift`: Given $\texttt{L}(\!|\texttt{foldr}\,(\Gamma, \to, p)|\!)$ and $\texttt{L}' = \texttt{v} :: \texttt{L}$, by Lemma 3.51 we get $\texttt{L}'(\!|\uparrow^1 \texttt{foldr}\,(\Gamma, \to, p)|\!)$, and by Lemma 3.48 that equals $\texttt{L}'(\!|\texttt{foldr}\,(\uparrow^1 \Gamma, \to, \uparrow^1 p)|\!)$.
4. `EqS`: Given $\texttt{L}(\!|Sx = Sy|\!)$, which equals $\texttt{S}\,(\texttt{L}(\!|x|\!)) = \texttt{S}\,(\texttt{L}(\!|y|\!))$, by inversion $\texttt{L}(\!|x|\!) = \texttt{L}(\!|y|\!)$ thus $\texttt{L}(\!|x = y|\!)$.
5. `SneqZ`: We can simplify $\texttt{L}(\!|\neg Sx = 0|\!)$ to $(\texttt{S}\,(\texttt{L}(\!|x|\!)) = 0) \to \texttt{False}$, which is true by discrimination on `nat`.
6. `SorZ`: By Lemma 3.50, $\texttt{L}(\!|x = 0 \vee \exists\,(\uparrow^1 x = Sv_1)|\!)$ reduces to
$$(\texttt{L}(\!|x|\!) = 0) + \sum_{v:\texttt{nat}} (\texttt{L}(\!|x|\!) = S\,(v))$$
By case analysis on $\texttt{L}(\!|x|\!)$: `nat`, this is clearly true.
7. `EqRefl`: $\texttt{L}(\!|x|\!) = \texttt{L}(\!|x|\!)$ is true by reflexivity.
8. `EqSubP`: Given $\texttt{L}(\!|x|\!) = \texttt{L}(\!|y|\!)$, this follows from Lemma 3.53.
9. `PlusZ`: $\texttt{L}(\!|x|\!) + 0 = \texttt{L}(\!|x|\!)$ is true by math.
10. `TimesZ`: $\texttt{L}(\!|x|\!) \times 0 = 0$ is true by math.
11. `PlusS`: $\texttt{L}(\!|x|\!) + \texttt{S}\,(\texttt{L}(\!|x|\!)) = \texttt{S}\,(\texttt{L}(\!|x|\!) + \texttt{L}(\!|x|\!))$ by math.
12. `TimesS`: $\texttt{L}(\!|x|\!) \times \texttt{S}\,(\texttt{L}(\!|x|\!)) = \texttt{L}(\!|x|\!) + (\texttt{L}(\!|x|\!) \times \texttt{L}(\!|x|\!))$ by math.

[2] Given $\texttt{L}(\!|p|\!)$, one can construct $[] \vdash^0 p$ as well. I have discovered a truly marvelous proof of this, which this footnote is too short to contain.

13. `ATrue`: $\mathtt{L}(\!|\top|\!) \triangleq \mathtt{True}$ is trivially true.
14. `AFalse`: Given $\mathtt{L}(\!|\bot|\!) \triangleq \mathtt{False}$, $\mathtt{L}(\!|p|\!)$ is true by explosion.
15. `ImpIntro`: Using Lemma 3.46, given the inductive hypothesis
$$\mathtt{L}(\!|\mathtt{foldr}\ (p :: \Gamma, \to, q)|\!) \triangleq \mathtt{L}(\!|p|\!) \times \mathtt{L}(\!|\Gamma|\!) \to \mathtt{L}(\!|q|\!)$$
we curry the function to get $\mathtt{L}(\!|\Gamma|\!) \to \mathtt{L}(\!|p|\!) \to \mathtt{L}(\!|q|\!)$.
16. `ImpElim`: If $\mathtt{L}(\!|p|\!) \to \mathtt{L}(\!|q|\!)$ and $\mathtt{L}(\!|p|\!)$, then we have $\mathtt{L}(\!|q|\!)$ by implication.
17. `NotIntro`: $\mathtt{L}(\!|p|\!) \to \mathtt{False}$ is equivalent to $\mathtt{L}(\!|p|\!) \to \mathtt{False}$.
18. `NotElim`: $\mathtt{L}(\!|p|\!) \to \mathtt{False}$ is equivalent to $\mathtt{L}(\!|p|\!) \to \mathtt{False}$.
19. `AndLeft`: Given $\mathtt{L}(\!|p|\!) \times \mathtt{L}(\!|q|\!)$, we have $\mathtt{L}(\!|p|\!)$ on the left.
20. `AndRight`: Given $\mathtt{L}(\!|p|\!) \times \mathtt{L}(\!|q|\!)$, we have $\mathtt{L}(\!|q|\!)$ on the right.
21. `AndIntro`: Assuming $\mathtt{L}(\!|p|\!)$ and $\mathtt{L}(\!|q|\!)$, we split and apply both assumptions to get $\mathtt{L}(\!|p|\!) \times \mathtt{L}(\!|q|\!)$.
22. `OrLeft`: $\mathtt{L}(\!|p|\!)$ is the left case of $\mathtt{L}(\!|p|\!) + \mathtt{L}(\!|q|\!)$.
23. `OrRight`: $\mathtt{L}(\!|q|\!)$ is the right case of $\mathtt{L}(\!|p|\!) + \mathtt{L}(\!|q|\!)$.
24. `OrElim`: Stumbling across $\mathtt{L}(\!|p|\!) + \mathtt{L}(\!|q|\!)$, $\mathtt{L}(\!|p|\!) \to \mathtt{L}(\!|r|\!)$, and $\mathtt{L}(\!|q|\!) \to \mathtt{L}(\!|r|\!)$, even a newborn logician can perform case analysis on the former to get $\mathtt{L}(\!|r|\!)$ by applying one of the latter two.
25. `Induct`: Upon being granted the gifts of base case $\mathtt{L}(\!|p\,(0)|\!)$ that by Lemma 3.56 is $(\mathtt{0} :: \mathtt{L})\ (\!|p|\!)$, as well as the inductive step $\mathtt{L}(\!|\forall\ (p \to \sigma^1 p)|\!)$ that by Lemma 3.57 is
$$\prod_{v:\mathtt{nat}} (\mathtt{v} :: \mathtt{L})\ (\!|p|\!) \to ((\mathtt{v} + \mathtt{1}) :: \mathtt{L})\ (\!|p|\!)$$
we easily get $\prod_{v:\mathtt{nat}} (\mathtt{v} :: \mathtt{L})\ (\!|p|\!)$ by induction on `v`.
26. `ForGen`: Assuming $\mathtt{L}(\!|\Gamma|\!)$ and $\mathtt{v} : \mathtt{nat}$, we would show $(\mathtt{v} :: \mathtt{L})(\!|p|\!)$ if we could apply the inductive hypothesis
$$(\mathtt{v} :: \mathtt{L})(\!|\uparrow^1 \Gamma|\!) \to (\mathtt{v} :: \mathtt{L})(\!|p|\!)$$
but deriving the antecedent takes some work. Fortunately, that work is done by Lemma 3.49!
27. `ForApp`: If you are walking down a lonely road and a hooded figure reveals to you in a hushed tone that $\prod_{v:\mathtt{nat}} (\mathtt{v} :: \mathtt{L})\ (\!|p|\!)$, do not be afraid! You can be secure in knowing $\mathtt{L}(\!|p\,(x)|\!)$ because by Lemma 3.56 it is equivalent to $(\mathtt{L}(\!|x|\!) :: \mathtt{L})\ (\!|p|\!)$, which is a special case of the hidden knowledge they have shared.
28. `ExElim`: If $\mathtt{L}(\!|\Gamma|\!) \to \sum_{v:\mathtt{nat}} (\mathtt{v} :: \mathtt{L})\ (\!|p|\!)$ and
$$(\mathtt{v} :: \mathtt{L})(\!|\uparrow^1 \Gamma|\!) \to (\mathtt{v} :: \mathtt{L})(\!|p|\!) \to (\mathtt{v} :: \mathtt{L})(\!|\uparrow^1 q|\!)$$
then to show $\mathtt{L}(\!|\Gamma|\!) \to \mathtt{L}(\!|q|\!)$ we can use Lemma 3.49 and then Lemma 3.51!
29. `Exhibit`: Hey, you want to hide x? I can do it, for a price. Hand over your $\mathtt{L}(\!|p\,(x)|\!)$ and I'll apply Lemma 3.56 to get $(\mathtt{L}(\!|x|\!) :: \mathtt{L})\ (\!|p|\!)$. That's not enough? Well, I know a guy who could get ya $\sum_{v:\mathtt{nat}} (\mathtt{v} :: \mathtt{L})\ (\!|p|\!)$, which I assure you is no different than the $\mathtt{L}(\!|\exists p|\!)$ you're looking for.
30. `LEM`: "I wish `LEM` need not have happened in my time," I hear you say. So do I, and so do all who live to see such times. But that is not for them to decide. All we have to decide is what to do with the time that is given us. If you thought that `constr` (`thm`), then I am glad you are here with me; here at the end of all things. Otherwise, it does not do to leave a live dragon out of your calculations, if you live near him: the mighty PALEM proudly proclaims $\mathtt{L}(\!|p \vee \neg p|\!)$! May the hair on your toes never fall out, dear reader.

Thus by induction we have $\mathtt{L}(\!|\Gamma|\!) \to \mathtt{L}(\!|p|\!)$ as claimed. □

The consistency of Peano Arithmetic immediately follows: Coq proves $\bot$ is not derivable from the axioms of PA.

Corollary 4.3. *There is no standard* `thm` $: [] \vdash^0 \bot$, *assuming* `constr` (`thm`) *or PALEM, given that Coq is consistent.*

4.2 Breast & Rewrite-Wing Combo

I show that nonstandard PA is equiconsistent with standard PA by showing how to convert proofs between them.

Theorem 4.4. *For a standard* `thm` $: \Gamma \vdash^d p$, *we have* $\Gamma^* \vdash^d p^*$ *with the same constructivity.*

Proof. Easily follows by induction on `thm` using the $*$-related Lemmas 3.11, 3.12, 3.13, 3.14, 3.16, 3.17, and 3.18. □

Now for the interesting direction:

Theorem 4.5. *For a nonstandard* `thm` $: \Gamma \vdash^d p$ *and all* $k > \wr\mathtt{thm}\wr$, *we have* $\lightning^k\Gamma \vdash^d \lightning^k p$ *with the same constructivity.*

Proof. I perform induction on `thm`, now for each nonstandard case in Table 1. I will omit all uses of conversion Lemmas 3.19 and 3.20 and uses of facts like that $\max(\wr\mathtt{thm}_1\wr, \wr\mathtt{thm}_2\wr) < k$ implies $\wr\mathtt{thm}_1\wr < k$ and $\wr\mathtt{thm}_2\wr < k$.

1. `NScopeLift`: Apply `ScopeLift`, using Lemma 3.22.
2. `NSorZ`: Apply `SorZ`, using Lemma 3.22.
3. `NEqSubP`: Apply `EqSubP`, using Lemma 3.24.
4. `NInduct`: Apply `Induct`, using L'Hôpital's rule[3].
5. `NForGen`: Apply `ForGen`, using Lemma 3.22.
6. `NForApp`: Apply `ForApp`, using Lemma 3.25.
7. `NExElim`: Apply `ExElim`, using Lemma 3.22.
8. `NExhibit`: Apply `Exhibit`, using Lemma 3.25.
9. `NLEM`: If we come across this case, the original proof was not constructive, so it's OK to apply `LEM`.
10. `Nbig`: Get your napkins ready for the main course. From Lemma 3.28, I need $\Gamma \vdash^d S^i 0 < S^k 0$ as a standard derivation from the inductive hypothesis $i < k$, which is achieved in Lemma 3.62.
11. In all other cases $\mathtt{N}\alpha$, apply α.

Thus by induction I have $\lightning^k\Gamma \vdash^d \lightning^k p$ as claimed. □

By these two Theorems 4.4 and 4.5, plus Lemma 3.29, I have formally-verified one of the most powerful principles underlying nonstandard methods.

Corollary 4.6 (Transfer Principle)**.** $\Gamma \vdash^d p$ *if and only if* $\Gamma^* \vdash^d p^*$.

The consistency of Nonstandard Peano Arithmetic immediately follows from Corollaries 4.6 and 4.3 as well.

Corollary 4.7. *There is no nonstandard* `thm` $: [] \vdash^0 \bot$, *assuming* `constr` (`thm`) *or PALEM, given that Coq is consistent.*

4.3 Drum & Thigh-orem Fill Up

When a nonstandard proposition does include $\Bbbk$, Theorem 4.5 provides a bottomless bowl of standard derivations for every substituted numeral $S^k 0$. I will now describe a novel and stronger method of transfer: a single derivation where $\Bbbk$ becomes a lower-bounded free variable within PA itself. Rather than replacing each use of an axiom `Nbig` (`i`, ...) with a concrete proof of $S^i 0 < S^k 0$, it is replaced with an application of `HypoUse`, as I will have inserted the hypothesis $S^i 0 < v_s$ (and a new variable v_s in place of $\Bbbk$) into the environment.

[3] Actually, just use Lemmas 3.25 and 3.26

Theorem 4.8. *Given any nonstandard* $\texttt{thm} : \Gamma \vdash^d p$ *and* $k = \wr\texttt{thm}\wr$*, we have the following standard derivation with the same constructivity:*

$$\texttt{push}\left(\curlywedge^{d+1}\Gamma, S^k 0 < v_{d+1}\right) \vdash^{d+1} \curlywedge^{d+1} p$$

Proof. I proved this by induction on `thm`. The only significant difference between this and Theorem 4.5 is that when recursing, we may need to rewrite $S^k 0 < v_{d+1}$ using Lemma 3.70. For example, we have $\wr\texttt{thm}\wr = \max\left(\wr\texttt{thm}_1\wr, \wr\texttt{thm}_2\wr\right)$ when there are two sub-derivations, thus their respective inductive hypotheses will have $S^{\wr\texttt{thm}_1\wr} 0 < v_{d+1}$ and $S^{\wr\texttt{thm}_2\wr} 0 < v_{d+1}$ instead. In these cases, I use Lemma 3.70 to rewrite the hypothesis according to Lemma 3.77. □

5. Country-Fried Coq

Given the formal verification of this technique for modeling nonstandard Peano arithmetic, I present the Country-Fried Coq language extension (alpha release) that adds support for a nonstandard $\Bbbk : \texttt{nat}$ and the infinite family of axioms $S^i 0 < \Bbbk$ to Coq, and a proof-of-concept preprocessor implementing it, the Deep Fryer.

In Country-Fried Coq, code may import `NonstandardNumber.v` to obtain the definition of $\texttt{nonstandard\_k} : \texttt{nat}$, and import any file of the form $\texttt{NonstandardAxiom}\,[x]\,\texttt{.v}$ to gain access to the axiom $\texttt{nonstandard\_axiom\_}\,[x] : [x] < \texttt{nonstandard\_k}$. For example, the command "`Require Import NonstandardAxiom7.`" imports the axiom $\texttt{nonstandard\_axiom\_7} : 7 < \texttt{nonstandard\_k}$. This nonstandard extension to the language cannot lead to `False`, as shown by Corollary 4.7, given that you trust Coq, otherwise you shouldn't be using it in the first place.

Deep Fryer Writing a preprocessor for this extension is rather simple. The Deep Fryer has two modes: a Nonstandard Mode (Algorithm 1) that creates an `Axiom` for $\Bbbk$ and each $x < \Bbbk$ used in your project, and a Standard Mode (Algorithm 2) that uses `Theorem` instead, baking-in a standard value for $\Bbbk$ based on the conversion from nonstandard $\Gamma \vdash^d p$ to standard $\lightning^k \Gamma \vdash^d \lightning^k p$ of Lemma 4.5.

I implement the Standard Mode proofs with Coq tactics that repeatedly apply $\texttt{lt\_n\_S} : \forall n \forall m. n < m \to S\ n < S\ m$ until the left-hand side is zero, then applies $\texttt{lt\_0\_succ} : \forall n. 0 < S\ n$ to finish, which always works because k was picked to be larger than every x. Compared to the proof of Lemma 3.62, this only takes two lines to write; producing the formal derivation is left as an exercise to the reader. I can't spoil all the fun for you! Once the preprocessor is complete, compilation can begin as normal.

Can I turn a `NPeanoTheorem` ***into a*** `Set` ***with*** `nonstandard_k`***?*** Not exactly. It's true that all the verification results of Section 4 transfer perfectly well to Country-Fried Coq. However, Country-Frying `Nbig` ("for all $i : \texttt{nat}$, $\Gamma \vdash^d S^i 0 < \Bbbk$") means that the Country-Fried $\Bbbk$ represents a nonstandardly nonstandard `nat` greater than any standardly nonstandard `nat`, thus the type of derivation $\texttt{Nbig}\,(\texttt{nonstandard\_k})$ is $\Gamma \vdash^d S^{\texttt{nonstandard\_k}} 0 < \Bbbk$, leading to double-fried, golden brown, extra-crispy goodness.

6. Takeout Menu

If you came for the computer science applications, this section is for you! Take home these freshly-cooked nuggets of research potential.

6.1 Asymptotic Analysis

As a computer scientist, you surely know Big O notation. Unlike other leading brands, nonstandard PA offers an equivalent yet superior formulation of this important research tool.

Algorithm 1 The Deep Fryer: Nonstandard Mode

1. Build a set X containing all numbers x such that "$\texttt{Require Import NonstandardAxiom}\,[x]$." is a command in the project.
2. Create a new file `NonstandardNumber.v` with this text:

```
Axiom nonstandard_k : nat.
```

3. For each $x \in X$, create a new file $\texttt{NonstandardAxiom}\,[x]\,\texttt{.v}$ by substituting x into this text:

```
Require Import NonstandardNumber.
Axiom nonstandard_axiom_[x] : [x] < nonstandard_k.
```

Algorithm 2 The Deep Fryer: Standard Mode

1. Build a set X like before (Algorithm 1).
2. Let $k = \max\left(X \cup \{0\}\right) + 1$.
3. Create a new file `NonstandardNumber.v` by substituting k into this text:

```
Definition nonstandard_k : nat := [k].
```

4. For each $x \in X$, create a new file $\texttt{NonstandardAxiom}\,[x]\,\texttt{.v}$ by substituting x into this text:

```
Require Import Coq.Arith.Arith.
Require Import NonstandardNumber.
Theorem nonstandard_axiom_[x] : [x] < nonstandard_k.
Proof.
repeat apply lt_n_S.
apply Nat.lt_0_succ.
Qed.
```

Definition 6.1. For $f, g : \mathbb{N} \to \mathbb{N}$, $g(n) = O(f(n))$ if and only if there exists $N, c \in \mathbb{N}$ such that for all $n > N$, $g(n) \leq c f(n)$.

Theorem 6.2. $g(n) = O(f(n))$ *if and only if* $g^*(\Bbbk) \leq c^* f^*(\Bbbk)$ *for some* $c \in \mathbb{N}$.

Proof. Clearly if for all $n > N$, $g(n) \leq c f(n)$, then by the transfer principle (4.6) and $\Bbbk > N^*$ we have $g^*(\Bbbk) \leq c^* f^*(\Bbbk)$. For the other direction, we can use Theorem 4.8 to show the following:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{thm} : \Gamma^* \vdash^d g^*(\Bbbk) \leq c^* \times f^*(\Bbbk)}{\texttt{push}\left(\Gamma, S^{\wr\texttt{thm}\wr} 0 < v_{d+1}\right) \vdash^{d+1} g(v_{d+1}) \leq c \times f(v_{d+1})}}{\Gamma \vdash^{d+1} S^{\wr\texttt{thm}\wr} 0 < v_{d+1} \to g(v_{d+1}) \leq c \times f(v_{d+1})}}{\Gamma \vdash^d \forall n. \left(S^{\wr\texttt{thm}\wr} 0 < n \to g(n) \leq c \times f(n)\right)}}{\Gamma \vdash^d \exists c. \forall n. \left(S^{\wr\texttt{thm}\wr} 0 < n \to g(n) \leq c \times f(n)\right)}}{\Gamma \vdash^d \exists N. \exists c. \forall n. \left(N < n \to g(n) \leq c \times f(n)\right)}}{\Gamma \vdash^d g(n) = O(f(n))}$$

□

As a practical example, $18(n + 500) + 1 = O(n)$ has been proven as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{9000 < \Bbbk}{9001 \leq \Bbbk}}{18\Bbbk + 9001 \leq 18\Bbbk + \Bbbk}}{18(\Bbbk + 500) + 1 \leq 19\Bbbk}}{18(n + 500) + 1 = O(n)}$$ [4]

[4] What? Nine thousand?!?

This technique applies more broadly than just asymptotic notation. If you are ever too lazy to choose a "sufficiently-large" number in your research, consider choosing the nonstandard number $\Bbbk$ instead. Ten minutes before the deadline, you can quickly use Theorem 4.5 to figure out a standard value to substitute for it.

6.2 Always-Satisfiable Constraint Solvers

Theorems 4.5 and 4.8 both can be thought of as constraint solvers on Peano Arithmetic proofs. From this innovative perspective, the "axioms" $\texttt{Nbig}\,(i, \ldots)$ may be viewed as constraints on the variable $\Bbbk$, and the equiconsistency of this nonstandard theory is a consequence of the fact that a system of only lower-bound-by-a-constant constraints is always solvable. If you ever have a system of always-satisfiable constraints, consider recasting your problem as a nonstandard theory.

6.3 Machine Learning Sum of Squares Randomized Quantum Cryptographic Big Data Science

If you work with real numbers, consider using Nonstandard Analysis, where you can use the infinitesimal $\frac{1}{\Bbbk} < \frac{1}{i}$ for all $i \in \mathbb{N}$ to simplify your limits, integrals, derivatives, and so on. I recommend [6] as an introduction to the field[5].

7. Surviving Thierry Coquand's Wild Ride

> "*A computer will do what you tell it to do, but that may be much different from what you had in mind.*" – Joseph Weizenbaum [3]

The trials and tribulations behind research papers are often swept under the rug and lost to time, but in this section I will reflect on some of the challenges I faced while formalizing these results in the roller-coaster ride currently known as Coq.

7.1 Minimum Scope

While writing this paper, it dawned on me that the "Minimum Scope" function actually has no purpose: that $\Gamma \vdash^d p$, aka $\texttt{PeanoTheorem}(d, \Gamma, p)$, has no functional reason to parameterize on d. It is only truly needed in Theorem 4.2, where the test $d \leq \texttt{length}\,(\texttt{L})$ ensures the list `L` is large enough to cover all free variable values, but this d value could easily be computed on-demand instead. This was disappointing because proving how the various transformation functions (Section 2.3) affect the "Minimum Scope" of a `PeanoExpr` or `PeanoProp` makes up a significant portion of the work.

My unapologetically post hoc excuse for this decision is that, by including d and proving that the deduction rules satisfy these constraints, we gain additional confidence that the deduction rules are properly tracking free variables, provide insight into how different syntactic operations affect DeBruijn Index values, and give future provers more control to restrict the number of free variables in their theorems. By the sunk cost principle, d shall never be removed.

7.2 Automation

For an amateur, it's hard to make reusable tactics to automate the many different kinds of inductive proofs done in this paper. Here were some of the most useful ones I wrote:

```
Inductive NatCompare (x y : nat) :=
| CompEq (e : x ?= y = Eq) (c : x = y)
| CompLt (e : x ?= y = Lt) (c : x < y)
| CompGt (e : x ?= y = Gt) (c : y < x).
Lemma nat_compare_split x y : NatCompare x y.
```

```
Ltac tactic_nat_cmp_split x y :=
let e := fresh "e" in
destruct(nat_compare_split x y) as [e|e|e];
rewrite e in *;
simpl in *;
auto.
```

```
Ltac tactic_nat_ltb_split_basic x y :=
let e := fresh "e" in
let LE := fresh "LE" in
let GT := fresh "GT" in
destruct(le_lt_dec (S x) y) as [LE|GT];
unfold Nat.ltb in *;
[ rewrite (leb_correct _ _ LE) in *
| rewrite (leb_correct_conv _ _ GT) in * ].
```

```
Ltac tactic_destruct_and := repeat match goal with
| [ H : ?P /\ ?Q |- _ ] => destruct H; auto
end.
```

7.3 Advice for You

1. Use the `Search` command to find uses of a given set of terms in your project, and `Locate` to find what file they're in.
2. Use library `Psatz` and its `lia` tactic (or similar) to prove basic arithmetic for you. Just don't check how the sausage is made...
3. Setoids are neat: the tactics `reflexivity`, `symmetry`, and `transitivity` worked with my `Set`-level if-and-only-if with no boilerplate after proving the properties normally.

   ```
   Iff (A B : Set) : Set := (A → B) × (B → A).
   ```

4. If you're stuck on an induction, generalize! For example, proving Lemma 3.14 for `PeanoProp` used induction to show $\forall p.\ \left(\forall d.\ \left(\uparrow^d p\right)^* = \uparrow^d p^*\right)$ rather than $\forall p.\ \left(\uparrow^d p\right)^* = \uparrow^d p^*$ for an arbitrary d.
5. To use multiple files, have a `_CoqProject` file that contains:

   ```
   -Q . ""
   ```

7.4 Statistics

This project hatched in May 2021, but it only grew its wings in December 2021. The Coq source files total to about 13K lines, 336K bytes, 54 Definitions, 12 Inductives, 16 Ltacs, 61 Fixpoints, 284 Lemmas, and 14 Theorems[6]. In comparison, the Deep Frier is just 70 simple lines of code I threw together one day in the D programming language.

8. Conclusion

I spent way too long verifying these delicious theorems for you, so I hope you licked your plate clean. Future work includes formalizing more-extreme nonstandard theories of arithmetic that include a $\texttt{standard}\,(\cdot)$ predicate for distinguishing between $\mathbb{N}$ and $\mathbb{N}^* \setminus \mathbb{N}$ within the theory itself.

9. Acknowledgments

Thanks to Giuseppe Peano for asking important questions like "How do you define a definition?" and then defining all the definitions for the standard natural numbers used in this paper.

In the spirit of being nonstandard, this paper was written using the LyX document preparation system. It was difficult to get SIGTBD's LaTeX class to work at first, but I was successful (it falls under "ACM SIGPLAN (Obsolete)"). LyX was very helpful for writing all these equations and derivations.

[5] The ordered field $(\mathbb{R}^*, +, \times, 0, 1, <)$, to be precise.

[6] Do you see what I mean by "Overly-Formalized" now?

References

[1] De bruijn index. *Wikipedia, the free encyclopedia*.

[2] Natural deduction. *Wikipedia, the free encyclopedia*.

[3] Quote. *CoolNSmart.com*.

[4] J. Avigad and J. Helzner. Transfer principles in nonstandard intuitionistic arithmetic. *Archive for Mathematical Logic*, 41(6):581–602, August 2002.

[5] Benno van den Berg, Eyvind Briseid, and Pavol Safarik. A functional interpretation for nonstandard arithmetic. *arXiv:1109.3103 [math]*, July 2012. arXiv: 1109.3103.

[6] Isaac Goldbring. *LECTURE NOTES ON NONSTANDARD ANALYSIS UCLA SUMMER SCHOOL IN LOGIC*. November 2014.

[7] Russell O'Connor. Essential Incompleteness of Arithmetic Verified by Coq. In *Theorem Proving in Higher Order Logics*, volume 3603, pages 245–260. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. Series Title: Lecture Notes in Computer Science.

[8] Giuseppe Peano. *Arithmetices principia: Nova methodo exposita*. Fratres Bocca, 1889.