Emily Riehl

Johns Hopkins University

A reintroduction to proofs

Plan

1. Logic, constructively

2. $\forall : \Pi :: \exists : \Sigma$

3. Peano's axioms, revisited

∞. $=$

1

Logic, constructively

Conjunction and disjunction

Forget truth tables! Instead, define the logical operators "and" $\wedge$ and "or" $\vee$ by:

Conjunction $\wedge$ is the logical operator defined by the rules:

- $^{\wedge}$intro: If p is true and q is true, then $p \wedge q$ is true.
- $^{\wedge}$elim$_1$: If $p \wedge q$ is true, then p is true.
- $^{\wedge}$elim$_2$: If $p \wedge q$ is true, then q is true.

Disjunction $\vee$ is the logical operator defined by the rules:

- $^{\vee}$intro$_1$: If p is true, then $p \vee q$ is true.
- $^{\vee}$intro$_2$: If q is true, then $p \vee q$ is true.
- $^{\vee}$elim: If $p \vee q$ is true, and if r can be derived from p and from q, then r is true.

Introduction rules explain how to prove a proposition involving a particular connective, while elimination rules explain how to use a hypothesis involving a particular connective.

Implication

Implication $\Rightarrow$ is the logical operator defined by the rules:

- $^{\Rightarrow}$intro: If q can be derived from the assumption that p is true, then $p \Rightarrow q$ is true.
- $^{\Rightarrow}$elim: If $p \Rightarrow q$ is true and p is true, then q is true.

Theorem. For any propositions p, q, and r, $((p \Rightarrow q) \wedge (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$.

Proof: By $^{\Rightarrow}$intro, assume that $(p \Rightarrow q) \wedge (q \Rightarrow r)$ is true; our goal is to prove $p \Rightarrow r$. By $^{\wedge}$elim$_1$ and $^{\wedge}$elim$_2$ it follows that $p \Rightarrow q$ and $q \Rightarrow r$ are true. By $^{\Rightarrow}$intro again, also assume p is true; now our goal is just to prove r. By $^{\Rightarrow}$elim, from p and $p \Rightarrow q$, we may conclude that q is true. By $^{\Rightarrow}$elim again, from q and $q \Rightarrow r$, we may conclude r is true as desired. □

givens:

$$p, q, r$$
$$(p \Rightarrow q) \wedge (q \Rightarrow r)$$
$$p \Rightarrow q$$
$$q \Rightarrow r$$
$$p$$
$$q$$
$$r$$

goal: $((p \Rightarrow q) \wedge (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$

Type theory

Type theory is a formal system for mathematical statements and proofs that has two primitive notions: types A, B and terms $a : A$, $b : B$.

In type theory, logic is unified with construction. In particular, some types are analogous to propositions while others are analogous to sets.

Mathematics in type theory:

- To state a conjecture, one forms a type that encodes its statement.
- To prove the theorem, one constructs a term in that type.

Given any types A and B, one may form the

product type $A \times B$, coproduct type $A + B$, function type $A \to B$

whose terms are governed by introduction and elimination (and computation) rules which extend the rules for conjunction, disjunction, and implication.

Conjunction and Products

Conjunction $\wedge$ is the logical operator defined by the rules:

- $^{\wedge}$intro: If p is true and q is true, then $p \wedge q$ is true.
- $^{\wedge}$elim$_1$: If $p \wedge q$ is true, then p is true.
- $^{\wedge}$elim$_2$: If $p \wedge q$ is true, then q is true.

Given types A and B, the product type $A \times B$ is governed by the rules:

- $^{\times}$intro: given terms $a : A$ and $b : B$ there is a term $(a, b) : A \times B$
- $^{\times}$elim$_1$: given a term $p : A \times B$ there is a term $\pi_1 p : A$
- $^{\times}$elim$_2$: given a term $p : A \times B$ there is a term $\pi_2 p : B$

plus computation rules that relate pairings and projections.

Implication and functions

Implication $\Rightarrow$ is the logical operator defined by the rules:

- ${}^{\Rightarrow}$intro: If q can be derived from the assumption that p is true, then $p \Rightarrow q$ is true.
- ${}^{\Rightarrow}$elim: If $p \Rightarrow q$ is true and p is true, then q is true.

Given types A and B, the function type $A \to B$ is governed by the rules:

- ${}^{\to}$intro: if given any term $x : A$ there is a term $b_x : B$,

  then there is a term $\lambda x.b_x : A \to B$
- ${}^{\to}$elim: given terms $f : A \to B$ and $a : A$, there is a term $f(a) : B$

plus computation rules that relate λ-abstractions and evaluations.

A proof/construction in type theory

The proof of transitivity of implication constructs the composition function:

Theorem. For any propositions p, q, and r, $((p \Rightarrow q) \wedge (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$.

Theorem. For any types P, Q, and R, $((P \to Q) \times (Q \to R)) \to (P \to R)$.

Construction: By $^{\to}$intro, suppose given $h : (P \to Q) \times (Q \to R)$; our goal is a term of type $P \to R$. By $^{\times}\text{elim}_1$ and $^{\times}\text{elim}_2$, we have $\pi_1 h : P \to Q$ and $\pi_2 h : Q \to R$. By $^{\to}$intro again, suppose given $p : P$; now our goal is a term of type R. By $^{\to}$elim, from $p : P$ and $\pi_1 h : P \to Q$, we obtain $\pi_1 h(p) : Q$. By $^{\to}$elim again, from $\pi_1 h(p) : Q$ and $\pi_2 h : Q \to R$, we obtain $\pi_2 h(\pi_1 h(p)) : R$ as desired. □

$$
\begin{array}{lr}
\text{givens:} & P, Q, R \\
 & h : (P \to Q) \times (Q \to R) \\
 & \pi_1 h : P \to Q \\
 & \pi_2 h : Q \to R \\
 & p : P \\
 & \pi_1 h(p) : Q \\
 & \pi_2 h(\pi_1 h(p)) : R \\
\hline
\multicolumn{2}{l}{\text{goal: } (P \to Q) \times (Q \to R) \to (P \to R)}
\end{array}
$$

This constructs a term $\lambda h.\lambda p.\pi_2 h(\pi_1 h(p)) : ((P \to Q) \times (Q \to R)) \to (P \to R)$.

Disjunction and coproducts

Disjunction $\vee$ is the logical operator defined by the rules:

- ${}^{\vee}\text{intro}_1$: If p is true, then $p \vee q$ is true.
- ${}^{\vee}\text{intro}_2$: If q is true, then $p \vee q$ is true.
- ${}^{\vee}\text{elim}$: If $p \vee q$ is true, and if r can be derived from p and from q, then r is true.

Given types A and B, the coproduct type $A + B$ is governed by the rules:

- ${}^{+}\text{intro}_1$: given a term $a : A$, there is a term $\iota_1 a : A + B$
- ${}^{+}\text{intro}_2$: given a term $b : B$, there is a term $\iota_2 b : A + B$
- ${}^{+}\text{elim}$: given a types C and terms $c_a, d_b : C$ for each $a : A$ and $b : B$ respectively, there is a term ${}^{+}\text{ind}(c, d)(x) : C$ for each $x : A + B$

plus computation rules that relate the inclusions and the elimination.

Another proof/construction in type theory

Theorem. For any types A, B, and C, $((A+B) \to C) \to ((A \to C) \times (B \to C))$.

Construction: By $^{\to}$intro, suppose given $h : (A+B) \to C$; our goal is a term of type $(A \to C) \times (B \to C)$. By $^{\times}$intro, it suffices to define terms of type $A \to C$ and type $B \to C$. By $^{\to}$intro, to define a term of type $A \to C$ it suffices to assume a term $a : A$ and define a term of type C. By $^{+}$intro$_1$, we then have a term $\iota_1 a : A + B$. Then by $^{\to}$elim we obtain a term $h(\iota_1 a) : C$. Similarly, by $^{\to}$intro, $^{+}$intro$_2$, and $^{\to}$elim we have $\lambda b.h(\iota_2 b) : B \to C$. $\square$

- $^{\to}$intro: if given any term $x : A$ there is a term $b_x : B$, there is a term $\lambda x.b_x : A \to B$
- $^{\times}$intro: given terms $a : A$ and $b : B$ there is a term $(a, b) : A \times B$
- $^{+}$intro$_1$: given a term $a : A$, there is a term $\iota_1 a : A + B$
- $^{\to}$elim: given terms $f : A \to B$ and $a : A$, there is a term $f(a) : B$

This constructs $\lambda h.(\lambda a.h(\iota_1 a), \lambda b.h(\iota_2 b)) : ((A+B) \to C) \to ((A \to C) \times (B \to C))$.

2

$$\forall : \Pi :: \exists : \Sigma$$

Universal and existential quantification

Let $p : X \to \{\perp, \top\}$ be an X-indexed family of propositions, a predicate $p(x)$ on $x \in X$.
For example:

- "$2^{2^n} - 1$ is prime" is a predicate on $n \in \mathbb{N}$
- "$z^2 = -1$" is a predicate on $z \in \mathbb{C}$

Universal quantification $\forall x \in X, p(x)$ is the logical formula defined by the rules:

- $^{\forall}$intro: If $p(x)$ can be derived from the assumption that x is an arbitrary element of X, then $\forall x \in X, p(x)$ is true.
- $^{\forall}$elim: If $\forall x \in X, p(x)$ is true and $a \in X$, then $p(a)$ is true.

Existential quantification $\exists x \in X, p(x)$ is the logical formula defined by the rules:

- $^{\exists}$intro: If $a \in X$ and $p(a)$ is true, then $\exists x \in X, p(x)$ is true.
- $^{\exists}$elim: If $\exists x \in X, p(x)$ is true and q can be derived from the assumption that $p(a)$ is true for some $a \in X$, then q is true.

Exchanging quantifiers

$^{\forall}$-intro: If $p(x)$ for any $x \in X$, then $\forall x \in X, p(x)$.
$^{\forall}$elim: If $\forall x \in X, p(x)$ and $a \in X$, then $p(a)$.

$^{\exists}$-intro: If $a \in X$ and $p(a)$, then $\exists x \in X, p(x)$.
$^{\exists}$elim: If $\exists x \in X, p(x)$ and q follows from $p(a)$ for some $a \in X$, then q.

Theorem. For any predicate $p(x, y)$ on $x \in X$ and $y \in Y$,

$$\exists y \in Y, \forall x \in X, p(x, y) \Rightarrow \forall x' \in X, \exists y' \in Y, p(x', y').$$

Proof: By $^{\Rightarrow}$intro, we may assume $\exists y \in Y, \forall x \in X, p(x, y)$; our goal is to prove $\forall x' \in X, \exists y' \in Y, p(x', y')$. By $^{\exists}$elim, we may assume $y_0 \in Y$ makes $\forall x \in X, p(x, y_0)$ true. By $^{\forall}$intro, we may fix $x' \in X$; our goal is to prove that $\exists y' \in Y, p(x', y')$. But by $^{\forall}$elim, we know that $p(x', y_0)$ is true. So by $^{\exists}$intro, it follows that $\exists y' \in Y, p(x', y')$ is true. □

givens:

goal: $\exists y \in Y, \forall x \in X, p(x, y)$ $\forall x' \in X$

Dependent type theory

Dependent type theory is a formal system for mathematical statements and proofs that, in addition to the types A, B and terms $a : A$, $b : B$, also has primitive notions of type families and term families that are indexed by previously-defined types.

Type families $B : A \to \mathrm{Type}$ are analogous to predicates and also to indexed families of sets, e.g.,

$$\text{is-prime} : \mathbb{N} \to \mathrm{Type},\ =_A : A \to A \to \mathrm{Type},\ \mathbb{R}^{\bullet} : \mathbb{N} \to \mathrm{Type},\ \mathrm{Mat}_{\bullet \times \bullet} : \mathbb{N} \to \mathbb{N} \to \mathrm{Type}$$

Term families $f : \prod_{x:A} B(x)$ are analogous to universal proofs or indexed families of elements and define dependent functions, e.g.,

$$\vec{0}^{\bullet} : \prod_{n:\mathbb{N}} \mathbb{R}^n\ ,\ \ I_{\bullet} : \prod_{n:\mathbb{N}} \mathrm{Mat}_{n,n}\ ,\ \ S_{\bullet} : \prod_{n:\mathbb{N}} \mathrm{Group}$$

Universal quantification and dependent functions

For any predicate $p : X \to \{\perp, \top\}$, the universal quantification $\forall x \in X, p(x)$ is the logical formula defined by the rules:

- $^{\forall}$intro: If $p(x)$ can be derived from the assumption that x is an arbitrary element of X, then $\forall x \in X, p(x)$ is true.
- $^{\forall}$elim: If $\forall x \in X, p(x)$ is true and $a \in X$, then $p(a)$ is true.

For any family of types $B : A \to \text{Type}$, the dependent function type $\prod_{x:A} B(x)$ is governed by the rules:

- $^{\Pi}$intro: if given any $x : A$ there is a term $b_x : B(x)$ there is a term $\lambda x.b_x : \prod_{x:A} B(x)$
- $^{\Pi}$elim: given terms $f : \prod_{x:A} B(x)$ and $a : A$ there is a term $f(a) : B(a)$

plus computation rules that relate λ-abstractions and evaluations.

For a constant type family $B : A \to \text{Type}$, the dependent function type recovers $A \to B$

Existential quantification and dependent sums

For any predicate $p : X \to \{\bot, \top\}$, the existential quantification $\exists x \in X, p(x)$ is the logical formula defined by the rules:

- $^{\exists}$intro: If $a \in X$ and $p(a)$ is true, then $\exists x \in X, p(x)$ is true.
- $^{\exists}$elim: If $\exists x \in X, p(x)$ is true and q can be derived from the assumption that $p(a)$ is true for some $a \in X$, then q is true.

For any family of types $B : A \to \mathrm{Type}$, the dependent sum type $\sum_{x:A} B(x)$ is governed by the rules:

- $^{\Sigma}$intro: if there are terms $a : A$ and $b : B(a)$, there is a term $(a, b) : \sum_{x:A} B(x)$
- $^{\Sigma}$elim: given a term $p : \sum_{x:A} B(x)$ there are terms $\pi_1 p : A$ and $\pi_2 p : B(\pi_1 p)$

plus computation rules that relate pairings and projections.

For a constant type family $B : A \to \mathrm{Type}$, the dependent sum type recovers $A \times B$.

Exchanging quantifiers, revisited

Theorem. For any $p(x, y)$, $\exists y \in Y, \forall x \in X, p(x, y) \Rightarrow \forall x' \in X, \exists y' \in Y, p(x', y')$.

Theorem. For any $P : X \to Y \to \mathrm{Type}$, $\Sigma_{y:Y}\Pi_{x:X}P(x, y) \to \Pi_{x':X}\Sigma_{y':Y}, P(x', y')$.

Proof: By ${}^{\Rightarrow}$intro, we may assume $\exists y \in Y, \forall x \in X, p(x, y)$; our goal is to prove $\forall x' \in X, \exists y' \in Y, p(x', y')$. By ${}^{\exists}$elim, we may assume $y_0 \in Y$ makes $\forall x \in X, p(x, y_0)$ true. By ${}^{\forall}$intro, we may fix $x' \in X$; our goal is to prove that $\exists y' \in Y, p(x', y')$. But by ${}^{\forall}$elim, we know that $p(x', y_0)$ is true. So by ${}^{\exists}$intro, it follows that $\exists y' \in Y, p(x', y')$ is true. $\square$

Proof: By ${}^{\to}$intro, we may assume $h : \Sigma_{y:Y}\Pi_{x:X}P(x, y)$; our goal is of type $\Pi_{x':X}\Sigma_{y':Y}P(x', y')$. By ${}^{\Sigma}$elim, we have $\pi_1 h : Y$ and $\pi_2 h : \Pi_{x:X}P(x, \pi_1 h)$. By ${}^{\Pi}$intro, we may fix $x' : X$; our goal is of type $\Sigma_{y':Y}P(x', y')$. But by ${}^{\Pi}$elim, we have $\pi_2 h(x') : P(x', \pi_1 h)$. So by ${}^{\Sigma}$intro, we then have $(\pi_1 h, \pi_2 h(x')) : \Sigma_{y':Y}P(x', y')$. $\square$

The constructs $\lambda h.\lambda x'.(\pi_1 h, \pi_2 h(x')) : \Sigma_{y:Y}\Pi_{x:X}P(x, y) \to \Pi_{x':X}\Sigma_{y':Y}, P(x', y')$.

3

Peano's axioms, revisited

The natural numbers

Dedekind's Categoricity Theorem. The natural numbers $\mathbb{N}$ are characterized by Peano's postulates:

- There is a natural number $0 \in \mathbb{N}$.
- Every natural number $n \in \mathbb{N}$ has a successor $\mathsf{suc}\, n \in \mathbb{N}$.
- 0 is not the successor of any natural number.
- No two natural numbers have the same successor.
- The principle of mathematical induction: for all predicates $P : \mathbb{N} \to \{\bot, \top\}$

$$P(0) \Rightarrow (\forall k \in \mathbb{N}, P(k) \Rightarrow P(\mathsf{suc}\, k)) \Rightarrow (\forall n \in \mathbb{N}, P(n))$$

A proof by induction

Theorem. For any $n \in \mathbb{N}$, $n^2 + n$ is even.

Proof: By induction on $n \in \mathbb{N}$:

- In the base case, when $n = 0$, $0^2 + 0 = 2 \times 0$, which is even.
- For the inductive step, assume for $k \in \mathbb{N}$ that $k^2 + k = 2 \times m$ is even. Then

$$\begin{aligned}(k+1)^2 + (k+1) &= (k^2 + k) + ((2 \times k) + 2) \\ &= (2 \times m) + (2 \times (k+1)) \\ &= 2 \times (m + k + 1) \qquad \text{is even.}\end{aligned}$$

By the principle of mathematical induction

$$\forall P, P(0) \Rightarrow (\forall k \in \mathbb{N}, P(k) \Rightarrow P(\mathsf{suc}\,k)) \Rightarrow (\forall n \in \mathbb{N}, P(n))$$

this proves that $n^2 + n$ is even for all $n \in \mathbb{N}$. $\square$

A construction by induction

The inductive proof not only demonstrates for all $n \in \mathbb{N}$ that $n^2 + n$ is even but also defines a function $m : \mathbb{N} \to \mathbb{N}$ so that $n^2 + n = 2 \times m(n)$.

Theorem. There is a function $m \colon \mathbb{N} \to \mathbb{N}$ so that $n^2 + n = 2 \times m(n)$ for all $n \in \mathbb{N}$.

Construction: By induction on $n \in \mathbb{N}$:

- In the base case, $0^2 + 0 = 2 \times 0$, so we define $m(0) \coloneqq 0$.
- For the inductive step, assume for $k \in \mathbb{N}$ that $k^2 + k = 2 \times m(k)$. Then

$$\begin{aligned}
(k+1)^2 + (k+1) &= (k^2 + k) + ((2 \times k) + 2) \\
&= (2 \times m(k)) + (2 \times (k+1)) \\
&= 2 \times (m(k) + k + 1)
\end{aligned}$$

so we define $m(k+1) \coloneqq m(k) + k + 1$.

By the principle of mathematical recursion, this defines a function $m : \mathbb{N} \to \mathbb{N}$ so that $n^2 + n = 2 \times m(n)$ for all $n \in \mathbb{N}$. $\square$

Induction and recursion

Recursion can be thought of as the constructive form of induction

$$\forall P, P(0) \Rightarrow (\forall k \in \mathbb{N}, P(k) \Rightarrow P(\mathsf{suc}k)) \Rightarrow (\forall n \in \mathbb{N}, P(n))$$

in which the predicate

$$P : \mathbb{N} \to \{\top, \bot\} \quad \text{such as} \quad P(n) :\equiv \exists m \in \mathbb{N}, n^2 + n = 2 \times m$$

is replaced by an arbitrary family of sets

$$P : \mathbb{N} \to \mathsf{Set} \quad \text{such as} \quad P(n) :\equiv \{m \in \mathbb{N} \mid n^2 + n = 2 \times m\}.$$

The output of a recursive construction is a dependent function $p \in \prod_{n \in \mathbb{N}} P(n)$ which specifies a value $p(n) \in P(n)$ for each $n \in \mathbb{N}$.

$$\forall P, (p_0 \in P(0)) \to (p_s \in \prod_{k \in \mathbb{N}} P(k) \to P(\mathsf{suc}k)) \to (p \in \prod_{n \in \mathbb{N}} P(n))$$

The recursive function $p \in \prod_{n \in \mathbb{N}} P(n)$ satisfies computation rules:

$$p(0) :\equiv p_0 \qquad p(\mathsf{suc}n) :\equiv p_s(n, p(n)).$$

The natural numbers in dependent type theory

The natural numbers type $\mathbb{N}$ is governed by the rules:

- ${}^{\mathbb{N}}$intro: there is a term $0 : \mathbb{N}$ and for any term $n : \mathbb{N}$ there is a term $\mathsf{suc}\, n : \mathbb{N}$

The elimination rule strengthens the principle of mathematical induction by replacing the predicate $P : \mathbb{N} \to \{\bot, \top\}$ by an arbitrary family of types $P : \mathbb{N} \to \mathsf{Type}$.

- ${}^{\mathbb{N}}$elim: for any type family $P : \mathbb{N} \to \mathsf{Type}$, to prove $p : \prod_{n:\mathbb{N}} P(n)$ it suffices to prove $p_0 : P(0)$ and $p_s : \prod_{k:\mathbb{N}} P(k) \to P(\mathsf{suc}\, k)$. That is

$$
{}^{\mathbb{N}}\mathsf{ind} : P(0) \to \left(\prod_{k \in \mathbb{N}} P(k) \to P(\mathsf{suc}\, k)\right) \to \left(\prod_{n \in \mathbb{N}} P(n)\right)
$$

Computation rules establish that p is defined recursively from p_0 and p_s.

Note the other two Peano postulates are missing because they are provable!

∞

$=$

Identity types

The following rules for identity types were developed by Martin-Löf:

Given a type A and terms $x, y : A$, the identity type $x =_A y$ is governed by the rules:

- $^{=}$intro: given a type A and term $x : A$ there is a term $\mathsf{refl}_x : x =_A x$

The elimination rule for the identity type defines an induction principle analogous to recursion over the natural numbers: it provides sufficient conditions for which to define a dependent function out of the identity type family.

- $^{=}$elim: for any type family $P(x, y, p)$ over $x, y : A$ and $p : x =_A y$, to prove $P(x, y, p)$ for all x, y, p it suffices to assume y is x and p is refl_x. That is

$$^{=}\mathsf{ind} : \left(\prod_{x:A} P(x, x, \mathsf{refl}_x)\right) \to \left(\prod_{x,y:A} \prod_{p:x=_A y} P(x, y, p)\right)$$

A computation rule establishes that the proof of $P(x, x, \mathsf{refl}_x)$ is the given one.

Symmetry and transitivity of identifications

${}^{=}$elim: For any type family $P(x, y, p)$ over $x, y : A$ and $p : x =_A y$,

$$ {}^{=}\mathrm{ind} : \Big(\prod_{x:A} P(x, x, \mathrm{refl}_x)\Big) \to \Big(\prod_{x,y:A} \prod_{p:x=_A y} P(x, y, p)\Big) $$

Theorem (symmetry). $(-)^{-1} : \prod_{x,y:A} x =_A y \to y =_A x$.

Construction: By ${}^{\Pi}$intro it suffices to assume $x, y : A$ and $p : x =_A y$ and then define a term of type $P(x, y, p) :\equiv y =_A x$. By ${}^{=}$elim, we may reduce to the case $P(x, x, \mathrm{refl}_x) :\equiv x =_A x$, for which we have $\mathrm{refl}_x : x =_A x$. □

Theorem (transitivity). $* : \prod_{x,y,z:A} x =_A y \to (y =_A z \to x =_A z)$.

Construction: By ${}^{\Pi}$intro it suffices to assume $x, y : A$ and $p : x =_A y$ and then define a term of type $Q(x, y, p) :\equiv \prod_{z:A} y =_A z \to x =_A z$. By ${}^{=}$elim, we may reduce to the case $Q(x, x, \mathrm{refl}_x) :\equiv \prod_{z:A} x =_A z \to x =_A z$, for which we have $\mathrm{id} :\equiv \lambda q.q : x =_A z \to x =_A z$. □

Functions preserve identifications

${}^{=}$elim: For any type family $P(x,y,p)$ over $x,y:A$ and $p:x=_A y$,

$$ {}^{=}\mathrm{ind} : \left(\prod_{x:A} P(x,x,\mathrm{refl}_x)\right) \to \left(\prod_{x,y:A} \prod_{p:x=_A y} P(x,y,p)\right) $$

In set theory, a function $f : X \to Y$ is well-defined: if $x = x'$ then $f(x) = f(x')$.

Theorem. For any $f : A \to B$ and $a, a' : A$, there is a term

$$ \mathrm{ap}_f : (a =_A a') \to (f(a) =_B f(a')). $$

Construction: Let $f : A \to B$. By ${}^{=}$elim applied to the family $P(x,y,p) :\equiv f(x) =_B f(y)$, to define $\mathrm{ap}_f : \prod_{a,a':A}(a =_A a') \to (f(a) =_B f(a'))$ we may reduce to the case $\prod_{a:A} f(a) =_B f(a)$, for which we have $\lambda a.\mathrm{refl}_{f(a)} : \prod_{a:A} f(a) =_B f(a)$. $\square$

Inductive constructions over the natural numbers

${}^{\mathbb{N}}$elim: For any type family $P(n)$ over $n : \mathbb{N}$,

$$
{}^{\mathbb{N}}\mathrm{ind} : P(0) \to \left(\prod_{k \in \mathbb{N}} P(k) \to P(\mathrm{suc}k)\right) \to \left(\prod_{n \in \mathbb{N}} P(n)\right)
$$

Using the elimination rule for the natural numbers type, (dependent) functions out of $\mathbb{N}$ may be defined inductively by specifying their values on 0 and $\mathrm{suc}k$ for any $k : \mathbb{N}$.

- $2\times : \mathbb{N} \to \mathbb{N}$ is defined by $\begin{cases} 2 \times 0 :\equiv 0 \\ 2 \times \mathrm{suc}k :\equiv \mathrm{suc}(\mathrm{suc}(2 \times k)) \end{cases}$
- $+ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is defined by $\begin{cases} m + 0 :\equiv m \\ m + \mathrm{suc}k :\equiv \mathrm{suc}(m + k) \end{cases}$
- $\mathrm{dist}_{2\times} : \prod_{m:\mathbb{N}} \prod_{n:\mathbb{N}} 2 \times m + 2 \times n =_{\mathbb{N}} 2 \times (m + n)$ is defined by

$$
\begin{cases} \mathrm{dist}_{2\times}(m, 0) :\equiv \mathrm{refl}_{2 \times m} \\ \mathrm{dist}_{2\times}(m, \mathrm{suc}k) :\equiv \mathrm{ap}_{\mathrm{suc} \circ \mathrm{suc}}(\mathrm{dist}_{2\times}(m, n)) \end{cases}
$$

A constructive proof revisited

We proved for any $n \in \mathbb{N}$, that $n^2 + n$ is even by induction and by recursively defining $m : \mathbb{N} \to \mathbb{N}$ so that $n^2 + n = 2 \times m(n)$.

Theorem. For $\text{square+self} : \mathbb{N} \to \mathbb{N}$ given by $\begin{cases} \text{square+self}(0) := 0 \\ \text{square+self}(\text{suc}\,k) := \\ \quad \text{square+self}(k) + 2 \times \text{suc}\,k \end{cases}$

$$\prod_{n:\mathbb{N}} \sum_{m:\mathbb{N}} \text{square+self}(n) =_{\mathbb{N}} 2 \times m.$$

Construction: By ${}^{\mathbb{N}}\text{elim}$, it suffices to prove two cases:

- For $0 : \mathbb{N}$, we have $(0, \text{refl}_0) : \sum_{m:\mathbb{N}} \text{square+self}(0) =_{\mathbb{N}} 2 \times m$.
- For $\text{suc}\,k : \mathbb{N}$, from $m(k) : \mathbb{N}$ and $p(k) : \text{square+self}(k) =_{\mathbb{N}} 2 \times m(k)$ we have:

$$\text{ap}_{+2\times\text{suc}\,k}\, p(k) : \text{square+self}(k) + 2 \times \text{suc}\,k =_{\mathbb{N}} 2 \times m(k) + 2 \times \text{suc}\,k$$

$$\text{dist}_{2\times}(m(k), 2 \times \text{suc}\,k) : 2 \times m(k) + 2 \times \text{suc}\,k =_{\mathbb{N}} 2 \times (m(k) + \text{suc}\,k)$$

Composing these identifications yields the desired term:

$$(m(k) + \text{suc}\,k, \text{ap}_{+2\times\text{suc}\,k}\, p(k) \cdot \text{dist}_{2\times}(m(k), 2 \times \text{suc}\,k)) : \sum_{m:\mathbb{N}} \text{square+self}(\text{suc}\,k) =_{\mathbb{N}} 2 \times m \quad \square$$

References

A reintroduction to proofs using introduction and elimination rules:

- Clive Newstead, An Infinite Descent into Pure Mathematics
  `https://infinitedescent.xyz/`

On dependent type theory and identity types (plus much more):

- Egbert Rijke, Introduction to Homotopy Type Theory,
  `arXiv:2212.11082` and forthcoming from *Cambridge University Press*

To explore computer formalization:

- Kevin Buzzard and Mohammad Pedramfar, The natural numbers game,
  `https://adam.math.hhu.de/#/`

Thank you!