



Emily Riehl

Johns Hopkins University

A reintroduction to proofs

Hardy Lecture Series (unofficial stop), Heilbronn Institute

Plan



1. Logic, constructively

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

3. Peano's axioms, revisited

$\infty.$ =



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 elim₁: If $p \wedge q$ is true, then p is true.
- \wedge elim₂: If $p \wedge q$ is true, then q is true.

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

- \vee intro₁: If p is true, then $p \vee q$ is true.
- \vee intro₂: 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 elim₁ and \wedge elim₂ 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. \square

givens:

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

$p \Rightarrow q$

$q \Rightarrow r$

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 (any terms can be identified) 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 $A \rightarrow B$

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 **elim₁**: If $p \wedge q$ is true, then p is true.
- \wedge **elim₂**: 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 **elim₁**: given a term $p : A \times B$ there is a term $\pi_1 p : A$
- \times **elim₂**: 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** $A \rightarrow B$ is governed by the rules:

- \rightarrow **intro**: if given any term $x : A$ there is a term $b_x : B$,
then there is a term $\lambda x. b_x : A \rightarrow B$
- \rightarrow **elim**: given terms $f : A \rightarrow B$ 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 , $((P \rightarrow Q) \times (Q \rightarrow R)) \rightarrow (P \rightarrow R)$.

Construction: By \rightarrow intro, suppose given $h : (P \rightarrow Q) \times (Q \rightarrow R)$; our goal is a term of type $P \rightarrow R$. By \times elim₁ and \times elim₂, we have $\pi_1 h : P \rightarrow Q$ and $\pi_2 h : Q \rightarrow R$. By \rightarrow intro again, suppose given $p : P$; now our goal is a term of type R . By \rightarrow elim, from $p : P$ and $\pi_1 h : P \rightarrow Q$, we obtain $\pi_1 h(p) : Q$. By \rightarrow elim again, from $\pi_1 h(p) : Q$ and $\pi_2 h : Q \rightarrow R$, we obtain $\pi_2 h(\pi_1 h(p)) : R$ as desired. \square

gives:

$h : (P \rightarrow Q) \times (Q \rightarrow R)$

$\pi_1 h : P \rightarrow Q$

$\pi_2 h : Q \rightarrow R$

$p : P$

$\pi_1 h(p) : Q$

$\pi_2 h(\pi_1 h(p)) : R$

goal: $((P \rightarrow Q) \times (Q \rightarrow R)) \rightarrow (P \rightarrow R)$

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

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 , $((A + B) \rightarrow C) \rightarrow ((A \rightarrow C) \times (B \rightarrow C))$.

Construction: By \rightarrow intro, suppose given $h : (A + B) \rightarrow C$; our goal is a term of type $(A \rightarrow C) \times (B \rightarrow C)$. By \times intro, it suffices to define terms of type $A \rightarrow C$ and type $B \rightarrow C$. By \rightarrow intro, to define a term of type $A \rightarrow C$ it suffices to assume a term $a : A$ and define a term of type C . By $+$ intro₁, we then have a term $\iota_1 a : A + B$. Then by \rightarrow elim we obtain a term $h(\iota_1 a) : C$, defining $\lambda a. h(\iota_1 a) : A \rightarrow C$. Similarly, by \rightarrow intro, $+$ intro₂, and \rightarrow elim we have $\lambda b. h(\iota_2 b) : B \rightarrow C$. \square

- \rightarrow intro: if given any term $x : A$ there is a term $b_x : B$, there is a term $\lambda x. b_x : A \rightarrow B$
- \times intro: given terms $a : A$ and $b : B$ there is a term $(a, b) : A \times B$
- $+$ intro₁: given a term $a : A$, there is a term $\iota_1 a : A + B$
- \rightarrow elim: given terms $f : A \rightarrow B$ and $a : A$, there is a term $f(a) : B$

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



2

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

Universal and existential quantification



Let $p : X \rightarrow \{\perp, \top\}$ 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, suppose given $x' \in X$; 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

givens: $\exists y \in Y, \forall x \in X, p(x, y)$
 y_0
 $\forall x \in X, p(x, y_0)$
 x'
 $p(x', y_0)$
 $\exists y' \in Y, p(x', y')$

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

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

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

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

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

Universal quantification and dependent functions



For any **predicate** $p : X \rightarrow \{\perp, \top\}$, 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** $B : A \rightarrow \text{Type}$, the **dependent function type** $\prod_{x:A} B(x)$ is governed by the rules:

- **Π intro**: if given any $x : A$ there is a term $b_x : B(x)$
there is a term $\lambda x. b_x : \prod_{x:A} B(x)$
- **Π 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 $B : A \rightarrow \text{Type}$, the dependent function type recovers $A \rightarrow B$

Existential quantification and dependent sums



For any **predicate** $p : X \rightarrow \{\perp, \top\}$, 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** $B : A \rightarrow \mathbf{Type}$, the **dependent pair type** $\sum_{x:A} B(x)$ is governed by the rules:

- Σ **intro**: if there are terms $a : A$ and $b : B(a)$, there is a term $(a, b) : \sum_{x:A} B(x)$
- Σ **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 $B : A \rightarrow \mathbf{Type}$, the dependent pair 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 $P : X \rightarrow Y \rightarrow \text{Type}$, $\Sigma_{y:Y} \Pi_{x:X} P(x, y) \rightarrow \Pi_{x':X} \Sigma_{y':Y} P(x', y')$.

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

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



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 $\text{suc } n \in \mathbb{N}$.
- 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 $P : \mathbb{N} \rightarrow \{\perp, \top\}$

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

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

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

By the principle of mathematical induction

$$\forall P, P(0) \Rightarrow (\forall k \in \mathbb{N}, P(k) \Rightarrow P(k+1)) \Rightarrow (\forall n \in \mathbb{N}, P(n))$$

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



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 $m : \mathbb{N} \rightarrow \mathbb{N}$ so that $n^2 + n = 2 \times m(n)$.

Theorem. There is a function $m : \mathbb{N} \rightarrow \mathbb{N}$ 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 $m(0) := 0$.
- 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 $m(k+1) := m(k) + k + 1$.

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

Induction and recursion



Recursion can be thought of as the constructive form of induction

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

in which the **predicate**

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

is replaced by an arbitrary **family of sets**

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

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 $n \in \mathbb{N}$.

$$\forall P, P(0) \rightarrow \left(\prod_{k \in \mathbb{N}} P(k) \rightarrow P(\text{suck}) \right) \rightarrow \prod_{n \in \mathbb{N}} P(n)$$

The recursive function $p \in \prod_{n \in \mathbb{N}} P(n)$ defined by p_0 and p_s satisfies **computation rules**:

$$p(0) := p_0 \quad p(\text{suc } n) := p_s(n, p(n)).$$

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 $\text{succ } n : \mathbb{N}$

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

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

$$\mathbb{N}_{\text{ind}} : P(0) \rightarrow \left(\prod_{k \in \mathbb{N}} P(k) \rightarrow P(\text{succ } k) \right) \rightarrow \prod_{n \in \mathbb{N}} P(n)$$

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 $\text{refl}_x : x =_A x$

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

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

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

Symmetry and transitivity of identifications



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

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

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

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

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

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

Functions preserve identifications



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

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

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

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

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

Construction: Let $f : A \rightarrow B$. By =elim applied to the family

$P(x, y, p) := f(x) =_B f(y)$, to define $\text{ap}_f : \prod_{a,a':A} (a =_A a') \rightarrow (f(a) =_B f(a'))$ we may reduce to the case $\prod_{a:A} f(a) =_B f(a)$, for which we have

$$\lambda a. \text{refl}_{f(a)} : \prod_{a:A} f(a) =_B f(a).$$



Inductive constructions over the natural numbers



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

$$\mathbb{N}\text{ind} : P(0) \rightarrow \left(\prod_{k \in \mathbb{N}} P(k) \rightarrow P(\text{suck}) \right) \rightarrow \prod_{n \in \mathbb{N}} P(n)$$

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 suck for any $k : \mathbb{N}$.

- $2 \times : \mathbb{N} \rightarrow \mathbb{N}$ is defined by
$$\begin{cases} 2 \times 0 := 0 \\ 2 \times \text{suck} := \text{suc}(\text{suc}(2 \times k)) \end{cases}$$
- $+: \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ is defined by
$$\begin{cases} m + 0 := m \\ m + \text{suck} := \text{suc}(m + k) \end{cases}$$
- $\text{dist}_{2 \times} : \prod_{m:\mathbb{N}} \prod_{n:\mathbb{N}} 2 \times m + 2 \times n =_{\mathbb{N}} 2 \times (m + n)$ is defined by
$$\begin{cases} \text{dist}_{2 \times}(m, 0) := \text{refl}_{2 \times m} \\ \text{dist}_{2 \times}(m, \text{suck}) := \text{ap}_{\text{suc} \circ \text{suc}}(\text{dist}_{2 \times}(m, n)) \end{cases}$$

A constructive proof revisited



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

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

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

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 $\text{suck} : \mathbb{N}$, from $m(k) : \mathbb{N}$ and $p(k) : \text{square+self}(k) =_{\mathbb{N}} 2 \times m(k)$ we have:

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

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

Composing these identifications yields the desired term:

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

References



A reintroduction to proofs using introduction and elimination rules:

- Clive Newstead, [An Infinite Descent into Pure Mathematics](#)

infinitedescent.xyz

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

- Egbert Rijke, [Introduction to Homotopy Type Theory](#), *Cambridge University Press*

[arXiv:2212.11082](https://arxiv.org/abs/2212.11082)

To explore computer formalization:

- ADAM: Anticipating the Digital Age of Mathematics, [The Lean Games Server](#),

adam.math.hhu.de

Thank you!