The Natural Numbers

Heshan Du
University of Nottingham Ningbo China

November 2024

Aims and Learning Objectives

- To be able to understand the meanings of the definitions of natural numbers, succ, pred, double, half, add, mul in Lean.
- To be able to apply the tactics *injection*, *induction*, *apply congr_arg succ* in Lean.
- To be able to construct proofs for tautologies about the natural numbers using Lean.

Reading

- Thorsten Altenkirch, *Introduction to Formal Reasoning*, 2023.
  - Chapter 6. The Natural Numbers

The Natural Numbers

- We have already used the natural numbers ($\mathbb{N}$) in examples. Now we will formally define them.
- We will in spirit follow Giuseppe Peano who codified the laws of natural numbers using predicate logic in the late 19th century. This is referred to as *Peano Arithmetic*.
- Peano viewed the natural numbers as created from *zero* (*0=zero*) and *succ* successor, i.e., $1 = succ\ 0$, $2 = succ\ 1$ and so on.
- How to define natural numbers in Lean? Note that $\mathbb{N} = nat$.

```
1 namespace nat
2
3 inductive nat : Type
4 | |
5 |
6
```

Basic Properties of $\mathbb{N}$

- Let us verify some basic properties of $\mathbb{N}$, which actually correspond to some of Peano's axioms.
- First, we want to verify that $0 \neq succ\ n$, which corresponds to $tt \neq ff$ for *bool*. How?

```
1 namespace nat
2
3 example : ∀ n : ℕ, 0 ≠ succ n :=
4 begin
5     |
6 end
7
8 end nat
```

succ is injective

- What is the meaning of ‘injective’ for a function?
- How to prove *succ* is injective?

```
1 namespace nat
2
3 theorem inj_succ : ∀ m n : nat, succ m = succ n → m = n :=
4 begin
5     |
6 end
7
8 end nat
```

Injection and Surjection

Definition

A function f is said to be one-to-one, or injective, if and only if $f(a) = f(b)$ implies that $a = b$ for all a and b in the domain of f. A function is said to be an injection if it is one-to-one.

Definition

A function f from A to B is called onto, or surjective, if and only if for every element $b \in B$ there is an element $a \in A$ with $f(a) = b$. A function f is called a surjection if it is onto.

The Tactic *injection*

- The tactic *injection* can be applied to a hypothesis of the form $h : succ\ m = succ\ n$.
- How to prove *succ* is injective using *injection*?

Structural Recursion

- You may have already seen recursive programs.
- When defining functions on $\mathbb{N}$, we will need recursion but unlike the general recursion available in programming languages, we will only use *structural recursion*.
- This is, when we define a function on the natural numbers, we can use the function on *n* to compute it for *succ n*.
- How to define the *double* function which doubles a number?

```
1 namespace nat
2
3 def double : ℕ → ℕ
4 |   |
5 |
6
7 end nat
```

Defining *half*

How to define the function *half* which halves a number?

```
1 namespace nat
2
3 def half : ℕ → ℕ
4 | |
5 |
6
7 end nat
```

Induction

- Proof by induction is very closely related to structural recursion which we have just seen, it is basically the same idea but for proofs.
- What is the meaning of ‘inverse’ for a function?
- How to prove that *half* is the inverse of *double*?

```
16 theorem half_double : ∀ n : ℕ, half (double n) = n :=
17 begin
18     |
19 end
```

Inverse

Definition

Let f be a one-to-one correspondence from the set A to the set B. The inverse function of f is the function that assigns to an element b belonging to B the unique element a in A such that $f(a) = b$. The inverse function of f is denoted by f^{-1}. Hence, $f^{-1}(b) = a$ when $f(a) = b$.

The Tactic *induction*

- The tactic *induction* works very similar to *cases*, but it gives us an extra assumption when proving the successor case.
- E.g., *induction n with n' ih*, where *ih* refers to *induction hypothesis*.

apply congr_arg succ

- To use the fact that *succ* preserves equality, we may *apply congr_arg succ*.
- Recall that the theorem *congr_arg* states that $\forall\ f\ :\ A \to B,\ \forall\ x\ y\ :A,\ x = y \to f\ x = f\ y$.

Addition

- While addition is an operation which you may have learnt already in kindergarden, it still needs to be defined.
- How to define *add* in Lean?

```
1 namespace ifr
2 open nat
3
4 def add : ℕ → ℕ → ℕ
5 | |
6 |
7
8 end ifr
```

- $m + n$ is defined as *add m n*.

Properties of Addition 1

- 0 is a *neutral element*.
- This is, $n + 0 = n$ and $0 + n = n$.
- How to prove $n + 0 = n$?

```
1 namespace ifr
2 open nat
3
4 theorem add_rneutr : ∀ n : ℕ, n + 0 = n :=
5 begin
6     |
7 end
8
9 end ifr
```

Properties of Addition 2

How to prove $0 + n = n$?

```
1 namespace ifr
2 open nat
3
4 theorem add_lneutr : ∀ n : ℕ, 0 + n = n :=
5 begin
6     |
7 end
8
9 end ifr
```

Properties of Addition 3

- Another important property of addition is that brackets do not matter.
- This is, $(l + m) + n = l + (m + n)$, which is called *associativity*.
- How to prove $\forall\ l\ m\ n\ :\ \mathbb{N},\ (l + m) + n = l + (m + n)$?

```
1 namespace ifr
2 open nat
3
4 theorem add_assoc : ∀ l m n : ℕ, (l + m) + n = l + (m + n) :=
5 begin
6     |
7 end
8
9 end ifr
```

Monoid

We have shown the following facts about $+$ and 0:

- 0 is right neutral : $n + 0 = n$ (add_rneutr),
- 0 is left neutral : $0 + n = n$ (add_lneutr),
- $+$ is associative: $(l + m) + n = l + (m + n)$ (add_assoc).

Such a structure is called a **monoid**.

Commutativity of Addition

How to prove $m + n = n + m$ in Lean?

```
1 namespace ifr
2 open nat
3
4 theorem add_comm : ∀ m n : ℕ , m + n = n + m :=
5 begin
6     |
7 end
8
9 end ifr
```

Commutative Monoid and Group

- Together with the previous facts, we have now shown that $\mathbb{N}$ with $+$ and 0 form a **commutative monoid**.
- Mathematicians prefer it, if you also have inverse as for the integers where for every integer i, there is $-i$ such that $i + (-i) = 0$ and $(-i) + i = 0$. Such a structure is called a **group**.

Multiplication

- How to define multiplication for numbers in Lean?

```
1 namespace ifr
2 open nat
3
4 def mul : ℕ → ℕ → ℕ
5 | |
6 |
7
8 end ifr
```

- $x * y$ is defined to stand for *mul x y*.
- As $+$ was repeated *succ*, $*$ is repeated $+$.
- This is, $m * n$ is m added n times.

Properties of Multiplication 1

How to prove the following theorems in Lean?

theorem mult_rneutr: $\forall\ n\ : \mathbb{N},\ n * 1 = n$

theorem mult_lneutr: $\forall\ n\ : \mathbb{N},\ 1 * n = n$

theorem mult_assoc : $\forall\ l\ m\ n\ : \mathbb{N},\ (l\ *\ m)\ *\ n = \ l\ *\ (m\ *\ n)$

theorem mult_comm: $\forall\ m\ n\ : \mathbb{N},\ m\ *\ n = n\ *\ m$

To prove the theorems above, you will certainly need to use the properties of addition, the order in which you prove the theorems may be different from the order above. You may want to use the properties in the next slide as well.

Properties of Multiplication 2

How to prove the following theorems in Lean?

mult_zero_l: $\forall\, n : \mathbb{N},\ 0 * n = 0$

mult_zero_r: $\forall\, n : \mathbb{N},\ n * 0 = 0$

mult_distr_r: $\forall\, l\ m\ n : \mathbb{N},\ l * (m + n) = l * m + l * n$

mult_distr_l: $\forall\, l\ m\ n : \mathbb{N},\ (m + n) * l = m * l + n * l$

Monoid

We have shown the following facts about $*$ and 1:

- 1 is right neutral: $n * 1 = n$ (mult_rneutr),
- 1 is left neutral: $1 * n = n$ (mult_lneutr),
- $*$ is associative: $(l * m) * n = l * (m * n)$ (mult_assoc).

Such a structure is called a *monoid*.

Commutative Monoid

Since $*$ is commutative, this is, $m * n = n * m$ (mult_comm), we have shown that $\mathbb{N}$ with $*$ and 1 form a *commutative monoid*.

Semiring

We have shown that:

- $+$ is associative: $(l + m) + n = l + (m + n)$ (add_assoc),
- $*$ is associative: $(l * m) * n = l * (m * n)$ (mult_assoc),
- Left and right distributivity:
  - $(m + n) * l = m * l + n * l$ (mult_distr_l) ;
  - $l * (m + n) = l * m + l * n$ (mult_distr_r),
- $+$ is commutative: $m + n = n + m$ (add_comm).

Therefore, $\mathbb{N}$ with $+$ and $*$ is a *semiring*. Since $*$ is commutative too, it is a **commutative semiring** .