Chapter 18: Natural Numbers in Agda

Zhenjiang Hu, Wei Zhang

School of Computer Science, PKU

November 19, 2025

Peano Natural Number

```
data ℕ : Set where
    zero : ℕ
    suc : ℕ → ℕ
```

0 = zero

1= suc zero

2 = suc (suc zero)

3= suc (suc (suc zero))

…

Some Operations on Natural Numbers

```
_+_ : ℕ → ℕ → ℕ
zero + n = n
suc m + n = suc (m + n)

_*_ : ℕ → ℕ → ℕ
zero * n = zero
suc m * n = n + (m * n)

pred : ℕ → ℕ
pred 0 = 0
pred (suc n) = n
```

Two Simple Theorems about Addition

```
0+ : ∀ (x : ℕ) → 0 + x ≡ x
0+ x = refl
```

```
+0 : ∀ (x : ℕ) → x + 0 ≡ x
+0 zero = refl
+0 (suc x) rewrite +0 x = refl
```

Associativity and Working with Holes

加法的结合律

```
+assoc : ∀ (x y z : ℕ) → x + (y + z) ≡ (x + y) + z
+assoc zero y z = refl
+assoc (suc x) y z rewrite +assoc x y z = refl
```

如何通过“hole”来交互式地开发以上的证明？

步骤1: 通过？ 引入hole

```
+assoc zero y z = ?
```

Ctrl+c Ctrl+l

```
+assoc zero y z = {! 0!}
```

Associativity and Working with Holes

加法的结合律

```
+assoc : ∀ (x y z : ℕ) → x + (y + z) ≡ (x + y) + z
+assoc zero y z = refl
+assoc (suc x) y z rewrite +assoc x y z = refl
```

如何通过“hole”来交互式地开发以上的证明?

步骤2: Ctrl+c Ctrl+, 观察正规化后的goal和 context

```
+assoc zero y z = {! 0!}
```

Goal:y+z ≡ y+z

----

z: N

y: N

Associativity and Working with Holes

加法的结合律

```
+assoc : ∀ (x y z : ℕ) → x + (y + z) ≡ (x + y) + z
+assoc zero y z = refl
+assoc (suc x) y z rewrite +assoc x y z = refl
```

如何通过“hole”来交互式地开发以上的证明?

步骤3: 输入解决方法，Ctrl+c Ctrl+r进行检查（有时可自动推导出解决方法）

```
+assoc zero y z = {! 0!}
```

```
+assoc zero y z = refl
```

Exercise: Prove the second case for +assoc.

Commutativity of Addition and Helper Lemmas

```
+comm : ∀ (x y : N) → x + y ≡ y + x
+comm zero y = ?
+comm (suc x) y = ?
```

Load file (Ctrl+c Ctrl+l)

```
+comm : ∀ (x y : N) → x + y ≡ y + x
+comm zero y = { }0
+comm (suc x) y = { }1


------------------------------------------------------------
?0 : zero + y ≡  y + zero
?1 : suc x + y ≡ y + suc x
```

Commutativity of Addition and Helper Lemmas

```
+comm : ∀ (x y : N) → x + y ≡ y + x
+comm zero y = { }0
```

观察hole (Ctrl+c Ctrl+,)

```
Goal: y ≡ y+0
--------------------------------------------------------
y: N
```

因此，

```
+comm : ∀ (x y : N) → x + y ≡ y + x
+comm zero y rewrite +0 y = refl
```

Commutativity of Addition and Helper Lemmas

```
+comm : ∀ (x y : N) → x + y ≡ y + x
+comm zero y rewrite +o y = refl
+comm (suc x) y = { }o
```

观察hole (Ctrl+c Ctrl+,)

```
Goal: suc (x + y) ≡ y + suc x
------------------------------------------------------------
y : N
x : N
```

我们可以利用归纳假设 x + y ≡ y + x 来重写上面的goal

Commutativity of Addition and Helper Lemmas

```
+comm : ∀ (x y : N) → x + y ≡ y + x
+comm zero y rewrite +o y = refl
+comm (suc x) y rewrite +comm x y = { }o
```

观察hole (Ctrl+c Ctrl+,)

```
Goal: suc (y + x) ≡ y + suc x
------------------------------------------------------------
y : N
x : N
```

证明辅助引理： suc (y + x) ≡ y + suc x ?

```
+suc : ∀ (x y : N) → x + (suc y) ≡ suc (x + y)
+suc zero y = refl
+suc (suc x) y rewrite +suc x y = refl
```

Commutativity of Addition and Helper Lemmas

```
+comm : ∀ (x y : N) → x + y ≡ y + x
+comm zero y rewrite +o y = refl
+comm (suc x) y rewrite +comm x y = { }o
```

使用辅助引理

```
+comm : ∀ (x y : N) → x + y ≡ y + x
+comm zero y rewrite +o y = refl
+comm (suc x) y rewrite +suc y x | +comm x y = refl
```

Distributivity of Multiplication and Choosing the Induction Variable

```
*distribr : ∀ (x y z : N) → (x + y) * z ≡ x * z + y * z
*distribr x y z = { }0
```

选择递归变量 Ctrl+c Ctrl+c

```
*distribr : ∀ (x y z : N) → (x + y) * z ≡ x * z + y * z
*distribr zero y z = { }0
*distribr (suc x) y z = { }1
```

Distributivity of Multiplication and Choosing the Induction Variable

```
*distribr : ∀ (x y z : N) → (x + y) * z ≡ x * z + y * z
*distribr zero y z = { }0
*distribr (suc x) y z = { }1
```

选择递归变量 Ctrl+c Ctrl+,

```
y * z ≡ y * z
```

Ctrl+c Ctrl+r

```
*distribr : ∀ (x y z : N) → (x + y) * z ≡ x * z + y * z
*distribr zero y z = refl
*distribr (suc x) y z = { }1
```

Distributivity of Multiplication and Choosing the Induction Variable

```
*distribr : ∀ (x y z : N) → (x + y) * z ≡ x * z + y * z
*distribr zero y z = refl
*distribr (suc x) y z = { }0
```

观察goal和context Ctrl+c Ctrl+,

```
z + (x + y) * z ≡ z + x * z + y * z
```

即： z + ((x + y) * z) ≡ (z + x * z) + y * z

```
*distribr : ∀ (x y z : N) → (x + y) * z ≡ x * z + y * z
*distribr zero y z = refl
*distribr (suc x) y z rewrite *distribr x y z = +assoc z (x * z) (y * z)
```

Arithmetic Comparison

```
_<_ : ℕ → ℕ → 𝔹
0 < 0 = ff
0 < (suc y) = tt
(suc x) < (suc y) = x < y
(suc x) < 0 = ff

_=ℕ_ : ℕ → ℕ → 𝔹
0 =ℕ 0 = tt
suc x =ℕ suc y = x =ℕ y
_ =ℕ _ = ff

_≤_ : ℕ → ℕ → 𝔹
x ≤ y = (x < y) || x =ℕ y

_>_ : ℕ → ℕ → 𝔹
a > b = b < a

_≥_ : ℕ → ℕ → 𝔹
a ≥ b = b ≤ a
```

Arithmetic Comparison

自然数不可能小于0

```
<-0 : ∀ (x : ℕ) → x < 0 ≡ ff
<-0 0 = refl
<-0 (suc y) = refl
```

传递性

```
<-trans : ∀ {x y z : ℕ} → x < y ≡ tt → y < z ≡ tt → x < z ≡ tt
<-trans {x} {0} p1 p2 rewrite <-0 x = 𝔹-contra p1
<-trans {0} {suc y} {0} p1 ()
<-trans {0} {suc y} {suc z} p1 p2 = refl
<-trans {suc x} {suc y} {0} p1 ()
<-trans {suc x} {suc y} {suc z} p1 p2 = <-trans {x} {y} {z} p1 p2
```

```
𝔹-contra : ff ≡ tt → ∀{ℓ} {P : Set ℓ} → P
```

Dotted Variables

```
<-trans : ∀ {x y z : N} →
            x < y ≡ tt → y < z ≡ tt → x < z ≡ tt
<-trans p q = { }0
```

观察goal和context Ctrl+c Ctrl+,

```
Goal: .x < .z ≡ tt
---------------------------------------------------
p : .x < .y ≡ tt
q : .y < .z ≡ tt
.z : N
.y : N
.x : N
```

Dotted Variables

```
<-trans : ∀ {x y z : N} →
            x < y ≡ tt → y < z ≡ tt → x < z ≡ tt
<-trans{x}{y}{z} p q = { }o
```

观察goal和context Ctrl+c Ctrl+,

```
Goal: x < z ≡ tt
------------------------------------------------
p : x < y ≡ tt
q : y < z ≡ tt
z : N
y : N
x : N
```

An Equality Test

A function f of type A ⟶ A ⟶ B is defined to be an equality test when f x y returns tt if and only if x ≡ y is provable.

如何证明 _=ℕ_ 是一个相等测试。

```
_=ℕ_ : ℕ → ℕ → 𝔹
0 =ℕ 0 = tt
suc x =ℕ suc y = x =ℕ y
_ =ℕ _ = ff
```

An Equality Test

A function f of type A ⟶ A ⟶ B is defined to be an equality test when f x y returns tt if and only if x ≡ y is provable.

```
=ℕ-to-≡ : ∀ {x y : ℕ} → x =ℕ y ≡ tt → x ≡ y
=ℕ-to-≡ {0} {0} u = refl
=ℕ-to-≡ {suc x} {0} ()
=ℕ-to-≡ {0} {suc y} ()
=ℕ-to-≡ {suc x} {suc y} u rewrite =ℕ-to-≡ {x} {y} u = refl
```

```
=ℕ-from-≡ : ∀ {x y : ℕ} → x ≡ y → x =ℕ y ≡ tt
=ℕ-from-≡ {x} refl = =ℕ-refl x

=ℕ-refl : ∀ (x : ℕ) → (x =ℕ x) ≡ tt
=ℕ-refl 0 = refl
=ℕ-refl (suc x) = =ℕ-refl x
```

Mutually Recursive Definitions

相互递归的函数定义

```
is-even : ℕ → 𝔹
is-odd : ℕ → 𝔹
is-even zero = tt
is-even (suc x) = is-odd x
is-odd zero = ff
is-odd (suc x) = is-even x
```

相互递归的证明

```
even~odd : ∀ (x : ℕ) → is-even x ≡ ~ is-odd x
odd~even : ∀ (x : ℕ) → is-odd x ≡ ~ is-even x
even~odd zero = refl
even~odd (suc x) = odd~even x
odd~even zero = refl
odd~even (suc x) = even~odd x
```

Homework

18.1. 证明 _*_ 满足交换性和结合律。

$\forall \{x\ y : N\} \rightarrow x * y \equiv y * x$

$\forall \{x\ y\ z : N\} \rightarrow x * (y * z) \equiv (x * y) * z$

18.2. 证明下面的性质。

```
∀ (n : ℕ) → n < n ≡ ff

∀ {x y : ℕ} → x < y ≡ tt → y < x ≡ ff

∀ (n m : ℕ) → n < m ≡ tt || n =ℕ m || m < n ≡ tt
```