



Reasoning About Lists

Principles of Functional Programming

Laws of Concat

Recall the concatenation operation `++` on lists.

We would like to verify that concatenation is associative, and that it admits the empty list `Nil` as neutral element to the left and to the right:

$$(xs \ ++ \ ys) \ ++ \ zs \ = \ xs \ ++ \ (ys \ ++ \ zs)$$

$$xs \ ++ \ Nil \ = \ xs$$

$$Nil \ ++ \ xs \ = \ xs$$

Q: How can we prove properties like these?

Laws of Concat

Recall the concatenation operation `++` on lists.

We would like to verify that concatenation is associative, and that it admits the empty list `Nil` as neutral element to the left and to the right:

$$(xs \ ++ \ ys) \ ++ \ zs \ = \ xs \ ++ \ (ys \ ++ \ zs)$$

$$xs \ ++ \ Nil \ = \ xs$$

$$Nil \ ++ \ xs \ = \ xs$$

Q: How can we prove properties like these?

A: By *structural induction* on lists.

Reminder: Natural Induction

Recall the principle of proof by *natural induction*:

To show a property $P(n)$ for all the integers $n \geq b$,

- ▶ Show that we have $P(b)$ (*base case*),
- ▶ for all integers $n \geq b$ show the *induction step*:
if one has $P(n)$, then one also has $P(n + 1)$.

Example

Given:

```
def factorial(n: Int): Int =  
  if n == 0 then 1          // 1st clause  
  else n * factorial(n-1)   // 2nd clause
```

Show that, for all $n \geq 4$

$\text{factorial}(n) \geq \text{power}(2, n)$

Base Case

Base case: 4

This case is established by simple calculations:

$$\text{factorial}(4) = 24 \geq 16 = \text{power}(2, 4)$$

Induction Step

Induction step: $n+1$

We have for $n \geq 4$:

$\text{factorial}(n + 1)$

Induction Step

Induction step: $n+1$

We have for $n \geq 4$:

$\text{factorial}(n + 1)$

$\geq (n + 1) * \text{factorial}(n)$ *// by 2nd clause in factorial*

Induction Step

Induction step: $n+1$

We have for $n \geq 4$:

$\text{factorial}(n + 1)$

$\geq (n + 1) * \text{factorial}(n)$ *// by 2nd clause in factorial*

$> 2 * \text{factorial}(n)$ *// by calculating*

Induction Step

Induction step: $n+1$

We have for $n \geq 4$:

`factorial(n + 1)`

`>= (n + 1) * factorial(n) // by 2nd clause in factorial`

`> 2 * factorial(n) // by calculating`

`>= 2 * power(2, n) // by induction hypothesis`

Induction Step

Induction step: $n+1$

We have for $n \geq 4$:

`factorial(n + 1)`

`>= (n + 1) * factorial(n) // by 2nd clause in factorial`

`> 2 * factorial(n) // by calculating`

`>= 2 * power(2, n) // by induction hypothesis`

`= power(2, n + 1) // by definition of power`

Referential Transparency

Note that a proof can freely apply reduction steps as equalities to some part of a term.

That works because pure functional programs don't have side effects; so that a term is equivalent to the term to which it reduces.

This principle is called *referential transparency*.

Structural Induction

The principle of structural induction is analogous to natural induction:

To prove a property $P(xs)$ for all lists xs ,

- ▶ show that $P(\text{Nil})$ holds (*base case*),
- ▶ for a list xs and some element x , show the *induction step*:
if $P(xs)$ holds, then $P(x :: xs)$ also holds.

Example

Let's show that, for lists xs , ys , zs :

$$(xs ++ ys) ++ zs = xs ++ (ys ++ zs)$$

To do this, use structural induction on xs . From the previous implementation of $++$,

```
extension [T](xs: List[T]
  def ++ (ys: List[T]) = xs match
    case Nil => ys
    case x :: xs1 => x :: (xs1 ++ ys)
```

distill two *defining clauses* of $++$:

```
Nil ++ ys = ys           // 1st clause
(x :: xs1) ++ ys = x :: (xs1 ++ ys) // 2nd clause
```

Base Case

Base case: Nil

For the left-hand side we have:

`(Nil ++ ys) ++ zs`

Base Case

Base case: Nil

For the left-hand side we have:

`(Nil ++ ys) ++ zs`

`= ys ++ zs` `// by 1st clause of ++`

Base Case

Base case: Nil

For the left-hand side we have:

`(Nil ++ ys) ++ zs`

`= ys ++ zs` *// by 1st clause of ++*

For the right-hand side, we have:

`Nil ++ (ys ++ zs)`

Base Case

Base case: Nil

For the left-hand side we have:

`(Nil ++ ys) ++ zs`

`= ys ++ zs // by 1st clause of ++`

For the right-hand side, we have:

`Nil ++ (ys ++ zs)`

`= ys ++ zs // by 1st clause of ++`

This case is therefore established.

Induction Step: LHS

Induction step: $x :: xs$

For the left-hand side, we have:

$((x :: xs) ++ ys) ++ zs$

Induction Step: LHS

Induction step: $x :: xs$

For the left-hand side, we have:

$((x :: xs) ++ ys) ++ zs$

$= (x :: (xs ++ ys)) ++ zs \quad // \text{ by 2nd clause of } ++$

Induction Step: LHS

Induction step: $x :: xs$

For the left-hand side, we have:

$((x :: xs) ++ ys) ++ zs$

$= (x :: (xs ++ ys)) ++ zs \quad // \text{ by 2nd clause of } ++$

$= x :: ((xs ++ ys) ++ zs) \quad // \text{ by 2nd clause of } ++$

Induction Step: LHS

Induction step: $x :: xs$

For the left-hand side, we have:

$((x :: xs) ++ ys) ++ zs$

$= (x :: (xs ++ ys)) ++ zs$ // by 2nd clause of ++

$= x :: ((xs ++ ys) ++ zs)$ // by 2nd clause of ++

$= x :: (xs ++ (ys ++ zs))$ // by induction hypothesis

Induction Step: RHS

For the right hand side we have:

$$(x :: xs) ++ (ys ++ zs)$$

Induction Step: RHS

For the right hand side we have:

$$(x :: xs) ++ (ys ++ zs)$$
$$= x :: (xs ++ (ys ++ zs)) \quad // \text{ by 2nd clause of } ++$$

So this case (and with it, the property) is established.

Exercise

Show by induction on xs that $xs ++ Nil = xs$.

How many equations do you need for the inductive step?

0 2

0 3

0 4

Exercise

Show by induction on xs that $xs ++ Nil = xs$.

How many equations do you need for the inductive step?

X 2

0 3

0 4