Reasoning About Lists
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 \( \text{Nil} \) as neutral element to the left and to the right:

\[
(xs \ ++ \ ys) \ ++ \ zs = xs \ ++ \ (ys \ ++ \ zs) \\
xs \ ++ \ \text{Nil} = xs \\
\text{Nil} \ ++ \ xs = xs
\]

**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:

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

Show that, for all $n \geq 4$

$$factorial(n) \geq power(2, n) = 2^n$$
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 \( n \geq 4 \):

\[
\text{factorial}(n + 1) \\
\geq (n + 1) \times \text{factorial}(n) \quad // \text{by 2nd clause in factorial}
\]
Induction Step

**Induction step**: \( n+1 \)

We have for \( n \geq 4 \):

\[
\text{factorial}(n + 1) \\
\geq (n + 1) \times \text{factorial}(n) \quad \text{// by 2nd clause in factorial} \\
> 2 \times \text{factorial}(n) \quad \text{// by calculating}
\]
**Induction Step**

**Induction step: n+1**

We have for $n \geq 4$:

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

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

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

$$\geq 2 \times \text{power}(2, n) \quad \text{// by induction hypothesis}$$
Induction Step

**Induction step: n+1**

We have for $n \geq 4$:

\[
\text{factorial}(n + 1)
\]

\[
\geq (n + 1) \times \text{factorial}(n) \quad \text{// by 2nd clause in factorial}
\]

\[
> 2 \times \text{factorial}(n) \quad \text{// by calculating}
\]

\[
\geq 2 \times \text{power}(2, n) \quad \text{// by induction hypothesis}
\]

\[
= \text{power}(2, n + 1) \quad \text{// by definition of power}
\]

\[
2 \times 2^n = 2^{n+1}
\]
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 `concat`,

```
def concat[T](xs: List[T], ys: List[T]) = xs match {
  case List() => ys
  case x :: xs1 => x :: concat(xs1, ys)
}
```

distill two defining clauses of `++`:

- $Nil \ ++ \ ys = ys$  \quad // 1st clause
- $(x :: xs1) \ ++ \ ys = x :: (xs1 \ ++ \ ys)$  \quad // 2nd clause
Block case: \texttt{Nil}

For the left-hand side we have:

\[(\texttt{Nil} \ ++ \ \texttt{ys}) \ ++ \ \texttt{zs}\]
Base Case

**Base case: Nil**

For the left-hand side we have:

$$(\text{Nil} \ ++ \ ys) \ ++ \ zs$$

$$= ys ++ zs \quad \text{// by 1st clause of ++}$$
Base Case

**Base case:** Nil

For the left-hand side we have:

\[(\text{Nil} \ ++ \ \text{ys}) \ ++ \ \text{zs}\]

\[= \ \text{ys} \ ++ \ \text{zs} \quad \quad \text{// by 1st clause of ++}\]

For the right-hand side, we have:

\[
\text{Nil} \ ++ (\text{ys} \ ++ \ \text{zs})
\]
Base Case

**Base case: Nil**

For the left-hand side we have:

\[(\text{Nil} ++ \text{ys}) ++ \text{zs}\]

\[= \text{ys} ++ \text{zs} \quad \text{// by 1st clause of ++}\]

For the right-hand side, we have:

\[\text{Nil} ++ (\text{ys} ++ \text{zs})\]

\[= \text{ys} ++ \text{zs} \quad \text{// by 1st clause of ++}\]

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:

\[
((x :: x) \++ y) \++ z = (x :: (x \++ y)) \++ z \quad // \text{by 2nd clause of ++}
\]
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:

\[
((x :: xs) ++ ys) ++ zs
\]

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

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

\[
= x :: (xs ++ (ys ++ zs)) \quad \text{// by induction hypothesis}
\]
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 \( xs \mathbin{++} \text{Nil} = xs \).

How many equations do you need for the inductive step?

\[
\begin{align*}
\text{Base case: } & \quad xs = \text{Nil} \\
& \quad \text{Nil} \mathbin{++} \text{Nil} \\
& = \quad \text{Nil} \quad \text{// by 1st clause} \\
\text{Induction step: } & \quad x :: xs \\
& \quad (x :: xs) \mathbin{++} \text{Nil} \\
& = \quad x :: (xs \mathbin{++} \text{Nil}) \quad \text{// 2nd clause} \\
& = \quad x :: xs \quad \text{// eq. i.h.}
\end{align*}
\]