Primitive recursion on lists:

\[
\begin{align*}
  f \; [] & = \text{base} \quad \text{-- base case} \\
  f \; (x : xs) & = \text{rec} \quad \text{-- recursive case}
\end{align*}
\]

- \text{base}: no call of \( f \)
- \text{rec}: only call(s) \( f \, \text{on} \; xs \)

\( f \) may have additional parameters.

Finding primitive recursive definitions

Example

\[
\text{concat} : \; [[a]] \rightarrow [a]
\]

Beyond primitive recursion: Multiple arguments

Example

\[
\text{zip} : \; [a] \rightarrow [b] \rightarrow [(a,b)]
\]
General recursion: Quicksort

Example

quicksort :: Ord a => [a] -> [a]
quicksort [] = []
quicksort (x:xs) =
    quicksort below ++ [x] ++ quicksort above

Example

quicksort :: Ord a => [a] -> [a]
quicksort [] = []
quicksort (x:xs) =
    quicksort below ++ [x] ++ quicksort above
    where
        below = [y | y <- xs, y <= x]
        above = [y | y <- xs, x < y]
Accumulating parameter

Idea: Result is accumulated in parameter and returned later

Example: list of all (maximal) ascending sublists in a list

\text{ups } [3, 0, 2, 3, 2, 4] =
Accumulating parameter

Idea: Result is accumulated in parameter and returned later
Example: list of all (maximal) ascending sublists in a list
ups $[3,0,2,3,2,4] = [[[3], [0,2,3], [2,4]]$

```haskell
ups2 :: Ord a => [a] -> [a] -> [[a]]
-- 1st param: input list
-- 2nd param: partial ascending sublist (reversed)
```
Idea: Result is accumulated in parameter and returned later

Example: list of all (maximal) ascending sublists in a list
ups \([3,0,2,3,2,4]\) = \([[3], [0,2,3], [2,4]]\)

\[
\begin{align*}
\text{ups2} & : \text{Ord} a \Rightarrow [a] \rightarrow [a] \rightarrow [[a]] \\
& \quad \text{1st param: input list} \\
& \quad \text{2nd param: partial ascending sublist (reversed)} \\
& \quad \text{ups2 (x:xs) (y:ys)} \\
& \quad \quad | x \geq y \quad = \text{ups2 xs (x:y:ys)} \\
& \quad \quad | \text{otherwise} \quad = \text{reverse (y:ys)} : \text{ups2 (x:xs) []} \\
& \quad \text{ups2 (x:xs) []} = \text{ups2 xs [x]} \\
& \text{ups2 [] ys} = [\text{reverse ys}] \\
\end{align*}
\]

\[
\begin{align*}
\text{ups : Ord a => [a] -> [[a]]} \\
\text{ups xs = ups2 xs []}
\end{align*}
\]
Identifiers of list type end in 's':
xs, ys, zs, ...

Example

\[
\text{even} :: \text{Int} \rightarrow \text{Bool} \\
\text{even } n = n == 0 || n > 0 && \text{odd} (n-1) || \text{odd}(n+1)
\]

\[
\text{odd} :: \text{Int} \rightarrow \text{Bool} \\
\text{odd } n = n /= 0 && (n > 0 && \text{even}(n-1) || \text{even}(n+1))
\]

Scoping by example

\[
x = y + 5 \\
y = x + 1 \text{ where } x = 7 \\
if y = y + x
\]

> f 3

\[
x = y + 5 \\
y = x + 1 \text{ where } x = 7 \\
if y = y + x
\]

> f 3
16
Scoping by example

\begin{verbatim}
x = y + 5
y = x + 1 where x = 7
if y = y + x
> i 3
16
\end{verbatim}

Binding occurrence

Bound occurrence

Scope of binding
Scoping by example

\[
x = y + 5
y = x + 1 \text{ where } x = 7
i \ y = y + x
\]

> i 3
16

**Binding occurrence**
**Bound occurrence**
**Scope of binding**

Scoping by example

\[
x = y + 5
y = x + 1 \text{ where } x = 7
i \ y = y + x
\]

> i 3
16

**Binding occurrence**
**Bound occurrence**
**Scope of binding**

Summary:
- Order of definitions is irrelevant
- Parameters and where-defs are local to each equation

5. Proofs
Guarantee functional (I/O) properties of software

- Testing can guarantee properties for *some* inputs.
- Mathematical proof can guarantee properties for *all* inputs.

*QuickCheck is good, proof is better*

*Beware of bugs in the above code;
I have only proved it correct, not tried it.*

Donald E. Knuth, 1977

5.1 Proving properties

What do we prove?

Equations $e_1 = e_2$
5.1 Proving properties

What do we prove?

Equations \( e_1 = e_2 \)

How do we prove them?

By using defining equations \( f \cdot p = t \)

A first, simple example

Remember: \[ \text{[] } + \text{ ys } = \text{ ys } \]
\[ (x:xs) + \text{ ys } = x : (xs + \text{ ys}) \]

Proof of \[ [1,2] + \text{ [] } = [1] + [2] \]:

\[ 1:2:[\text{[] } + \text{ [] }] \]
A first, simple example

Remember: \[ \text{[] ++ ys = ys} \]  
\[(x:xs) ++ ys = x : (xs ++ ys)\]

Proof of \[[1,2] ++ [] = [1] ++ [2]:\]
1:2:[] ++ []
= 1 : (2:[] ++ [])  -- by def of ++

A first, simple example

Remember: \[ \text{[] ++ ys = ys} \]  
\[(x:xs) ++ ys = x : (xs ++ ys)\]

Proof of \[[1,2] ++ [] = [1] ++ [2]:\]
1:2:[] ++ []
= 1 : (2:[] ++ [])  -- by def of ++

A first, simple example

Remember: \[ \text{[] ++ ys = ys} \]  
\[(x:xs) ++ ys = x : (xs ++ ys)\]

Proof of \[[1,2] ++ [] = [1] ++ [2]:\]
1:2:[] ++ []
= 1 : (2:[] ++ [])  -- by def of ++

A first, simple example

Remember: \[ \text{[] ++ ys = ys} \]  
\[(x:xs) ++ ys = x : (xs ++ ys)\]

Proof of \[[1,2] ++ [] = [1] ++ [2]:\]
1:2:[] ++ []
= 1 : (2:[] ++ [])  -- by def of ++

= 1 : 2 : []  -- by def of ++
= 1 : ([] ++ 2::[])  -- by def of ++
A first, simple example

Remember:

\[
[x:] + y = y \\
(x:x) + y = x : (x + y)
\]

Proof of \([1,2] + [] = [1] + [2]\):

\[
\begin{align*}
1:2:[] + [] &= 1 : (2:[] + []) \quad \text{-- by def of } + \\
1:2 : ([] + []) &= 1 : 2 : ([] + []) \quad \text{-- by def of } + \\
1 : 2 : [] &= 1 : 2 : [] \quad \text{-- by def of } + \\
1 : ([] + 2 : []) &= 1 : ([] + 2 : []) \quad \text{-- by def of } + \\
1 : [] + 2 : [] &= 1 : [] + 2 : [] \quad \text{-- by def of } + \\
\end{align*}
\]

Observation: first used equations from left to right (ok), then from right to left (strange!)

A more natural proof of \([1,2] + [] = [1] + [2]\):

\[
\begin{align*}
1:2:[] + [] &= 1 : (2:[] + []) \quad \text{-- by def of } + \\
1:2 : ([] + []) &= 1 : 2 : ([] + []) \quad \text{-- by def of } + \\
1 : 2 : [] &= 1 : 2 : [] \quad \text{-- by def of } + \\
1 : ([] + 2 : []) &= 1 : ([] + 2 : []) \quad \text{-- by def of } + \\
1 : [] + 2 : [] &= 1 : [] + 2 : [] \quad \text{-- by def of } + \\
\end{align*}
\]
A more natural proof of \([1,2] \ ++ \ [] = [1] \ ++ [2]\):

\[
\begin{align*}
1:2:[] &\ ++ \ [] \\
= 1 : (2:[] \ ++ \ []) &\quad --\ by\ def\ of\ ++ \\
= 1 : 2 : ([] \ ++ \ []) &\quad --\ by\ def\ of\ ++ \\
= 1 : 2 : [] &\quad --\ by\ def\ of\ ++
\end{align*}
\]

Proofs of \(e_1 = e_2\) are often better presented as two reductions to some expression \(e\):

\[
\begin{align*}
e_1 &= \ldots = e \\
e_2 &= \ldots = e
\end{align*}
\]

**Fact** If an equation does not contain any variables, it can be proved by evaluating both sides separately and checking that the result is identical.

But how to prove equations with variables, for example

*associativity* of \(\++\):

\[
(xs \ ++ \ ys) \ ++ \ zs = xs \ ++ \ (ys \ ++ \ zs)
\]
Properties of recursive functions are proved by induction

Induction on natural numbers: see Diskrete Strukturen

Induction on lists: here and now

Structural induction on lists

To prove property $P(xs)$ for all finite lists $xs$

Base case: Prove $P([], \ldots)$. and
Structural induction on lists

To prove property $P(xs)$ for all finite lists $xs$

Base case: Prove $P([])$ and

Induction step: Prove $P(xs)$ implies $P(x:xs)$

This is called structural induction on $xs$. 
Structural induction on lists

To prove property $P(xs)$ for all finite lists $xs$

Base case: Prove $P([])$ and

Induction step: Prove $P(xs)$ implies $P(x:xs)$

This is called structural induction on $xs$.
It is a special case of induction on the length of $xs$.

Example: associativity of ++

Lemma $app\_assoc: (xs ++ ys) ++ zs = xs ++ (ys ++ zs)$
Proof by structural induction on $xs$

Base case:
To show: $([], ys) ++ zs = [] ++ (ys ++ zs)$

Example: associativity of ++

Lemma $app\_assoc: (xs ++ ys) ++ zs = xs ++ (ys ++ zs)$
Proof by structural induction on $xs$
Example: associativity of ++

**Lemma** `app_assoc`: \((xs \mathbin{++} ys) \mathbin{++} zs = xs \mathbin{++} (ys \mathbin{++} zs)\)

**Proof** by structural induction on `xs`

**Base case:**

To show: \(([] \mathbin{++} ys) \mathbin{++} zs = [] \mathbin{++} (ys \mathbin{++} zs)\)

\(([] \mathbin{++} ys) \mathbin{++}zs = ys \mathbin{++} zs\)

= `ys ++ zs`  \hspace{1em} -- by def of `++`

= `[] ++ (ys ++ zs)`  \hspace{1em} -- by def of `++`

Example: associativity of ++

**Lemma** `app_assoc`: \((xs \mathbin{++} ys) \mathbin{++} zs = xs \mathbin{++} (ys \mathbin{++} zs)\)

**Proof** by structural induction on `xs`

**Base case:**

To show: \(([] \mathbin{++} ys) \mathbin{++} zs = [] \mathbin{++} (ys \mathbin{++} zs)\)

\(([] \mathbin{++} ys) \mathbin{++} zs = ys \mathbin{++} zs\)

= `ys ++ zs`  \hspace{1em} -- by def of `++`

= `[] ++ (ys ++ zs)`  \hspace{1em} -- by def of `++`

**Induction step:**

To show: \(((x:xs) \mathbin{++} ys) \mathbin{++} zs = (x:xs) \mathbin{++} (ys \mathbin{++} zs)\)
Example: associativity of ++

**Lemma** app.assoc: \((xs ++ ys) ++ zs = xs ++ (ys ++ zs)\)

**Proof** by structural induction on \(xs\)

Base case:
To show: \(([] ++ ys) ++ zs = [] ++ (ys ++ zs)\)

\(([] ++ ys) ++ zs\)

\(= ys ++ zs \quad \text{-- by def of ++}\)

\(= [] ++ (ys ++ zs) \quad \text{-- by def of ++}\)

Induction step:
To show: \(((x:xs) ++ ys) ++ zs = (x:xs) ++ (ys ++ zs)\)

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

\(= (x : (xs ++ ys)) ++ zs \quad \text{-- by def of ++}\)

\(= x : ((xs ++ ys) ++ zs) \quad \text{-- by def of ++}\)
Example: associativity of ++

**Lemma** app_assoc: \((xs ++ ys) ++ zs = xs ++ (ys ++ zs)\)

**Proof** by structural induction on \(xs\)

Base case:
To show: \(([] ++ ys) ++ zs = [] ++ (ys ++ zs)\)
\[([] ++ ys) ++ zs = ys ++ zs \quad \text{-- by def of ++}\]
\[= [] ++ (ys ++ zs) \quad \text{-- by def of ++}\]

Induction step:
To show: \(((x:xs) ++ ys) ++ zs = (x:xs) ++ (ys ++ zs)\)
\[((x:xs) ++ ys) ++ zs = (x : (xs ++ ys)) ++ zs \quad \text{-- by def of ++}\]
\[= x : ((xs ++ ys) ++ zs) \quad \text{-- by def of ++}\]
\[= x : (xs ++ (ys ++ zs)) \quad \text{-- by IH}\]
\[(x:xs) ++ (ys ++ zs) \quad \text{-- by def of ++}\]

---

**Lemma** \(P(xs)\)

**Proof** by structural induction on \(xs\)

Base case:
To show: \(P([])\)
**Lemma** $P(xs)$

**Proof** by structural induction on $xs$

Base case:
To show: $P([])$

*Proof of $P([])$*

**Example: length of ++**

**Lemma** $\text{length}(xs ++ ys) = \text{length} \; xs + \text{length} \; ys$

**Proof** by structural induction on $xs$
Example: length of ++

**Lemma** \( \text{length}(\text{x}:\text{xs} + \text{ys}) = \text{length}(\text{xs}) + \text{length}(\text{ys}) \)

**Proof** by structural induction on \( \text{xs} \)

**Base case:**

To show: \( \text{length}([]) + \text{length}(\text{ys}) = \text{length}(\text{ys}) \)  
\[ = \text{length}(\text{ys}) \quad -- \text{by def of ++} \]
\[ = 0 + \text{length}(\text{ys}) \quad -- \text{by def of length} \]
\[ = \text{length}(\text{ys}) \]

**Induction step:**

To show: \( \text{length}((\text{x}:\text{xs}) + \text{ys}) = \text{length}(\text{xs}) + \text{length}(\text{ys}) \)

\[ = \text{length}(\text{x} : (\text{xs} + \text{ys})) \quad -- \text{by def of ++} \]
Example: length of ++

**Lemma** \( \text{length}(\text{x} ++ \text{y}) = \text{length} \text{x} + \text{length} \text{y} \)

**Proof** by structural induction on \( \text{x} \)

**Base case:**
To show: \( \text{length} ([] ++ \text{y}) = \text{length} [] + \text{length} \text{y} \)

\[
\begin{align*}
\text{length} [] + \text{length} \text{y} \\
= \text{length} \text{y} & \quad \text{by def of ++} \\
0 + \text{length} \text{y} & \quad \text{by def of length} \\
= \text{length} \text{y}
\end{align*}
\]

**Induction step:**
To show: \( \text{length}((\text{x}:\text{x})++\text{y}) = \text{length}(\text{x}:\text{x}) + \text{length} \text{y} \)

\[
\begin{align*}
\text{length}(\text{x}:\text{x}) + \text{length} \text{y} \\
= \text{length}(\text{x} :: (\text{x} ++ \text{y})) & \quad \text{by def of ++} \\
= 1 + \text{length}(\text{x} ++ \text{y}) & \quad \text{by def of length}
\end{align*}
\]

Example: length of ++

**Lemma** \( \text{length}(\text{x} ++ \text{y}) = \text{length} \text{x} + \text{length} \text{y} \)

**Proof** by structural induction on \( \text{x} \)

**Base case:**
To show: \( \text{length} ([] ++ \text{y}) = \text{length} [] + \text{length} \text{y} \)

\[
\begin{align*}
\text{length} [] + \text{length} \text{y} \\
= \text{length} \text{y} & \quad \text{by def of ++} \\
0 + \text{length} \text{y} & \quad \text{by def of length} \\
= \text{length} \text{y}
\end{align*}
\]

**Induction step:**
To show: \( \text{length}((\text{x}:\text{x})++\text{y}) = \text{length}(\text{x}:\text{x}) + \text{length} \text{y} \)

\[
\begin{align*}
\text{length}(\text{x}:\text{x}) + \text{length} \text{y} \\
= \text{length}(\text{x} :: (\text{x} ++ \text{y})) & \quad \text{by def of ++} \\
= 1 + \text{length}(\text{x} ++ \text{y}) & \quad \text{by def of length}
\end{align*}
\]

Example: length of ++

**Lemma** \( \text{length}(\text{x} ++ \text{y}) = \text{length} \text{x} + \text{length} \text{y} \)

**Proof** by structural induction on \( \text{x} \)

**Base case:**
To show: \( \text{length} ([] ++ \text{y}) = \text{length} [] + \text{length} \text{y} \)

\[
\begin{align*}
\text{length} [] + \text{length} \text{y} \\
= \text{length} \text{y} & \quad \text{by def of ++} \\
0 + \text{length} \text{y} & \quad \text{by def of length} \\
= \text{length} \text{y}
\end{align*}
\]

**Induction step:**
To show: \( \text{length}((\text{x}:\text{x})++\text{y}) = \text{length}(\text{x}:\text{x}) + \text{length} \text{y} \)

\[
\begin{align*}
\text{length}(\text{x}:\text{x}) + \text{length} \text{y} \\
= \text{length}(\text{x} :: (\text{x} ++ \text{y})) & \quad \text{by def of ++} \\
= 1 + \text{length}(\text{x} ++ \text{y}) & \quad \text{by def of length}
\end{align*}
\]
Example: reverse of ++

Lemma \( \text{reverse}(xs ++ ys) = \text{reverse} ys ++ \text{reverse} xs \)
Proof by structural induction on \( xs \)

Base case:

Example: reverse of ++

Lemma \( \text{reverse}(xs ++ ys) = \text{reverse} ys ++ \text{reverse} xs \)
Proof by structural induction on \( xs \)

Base case:
To show: \( \text{reverse} ([] ++ ys) = \text{reverse} ys ++ \text{reverse} [] \)
\[ \begin{align*}
    \text{reverse} ([] ++ ys) &= \text{reverse} ys \quad \text{-- by def of ++} \\
    \text{reverse} ys ++ \text{reverse} [] &= \text{reverse} ys ++ [] \quad \text{-- by def of reverse} \\
    &= \text{reverse} ys \quad \text{-- by ...}
\end{align*} \]
**Example: reverse of ++**

**Lemma** \(\text{reverse}(xs ++ ys) = \text{reverse} ys ++ \text{reverse} xs\)

**Proof** by structural induction on \(xs\)

Base case:
To show: \(\text{reverse} ([] ++ ys) = \text{reverse} ys ++ \text{reverse} []\)

\(\text{reverse} ([] ++ ys) = \text{reverse} ys\) \hspace{1cm} -- by def of ++
\n\(\text{reverse} ys ++ \text{reverse} []\) \hspace{1cm} -- by def of reverse
\n\(= \text{reverse} ys\) \hspace{1cm} -- by Lemma app.Nil2

**Lemma app.Nil2**: \(xs ++ [] = xs\)

**Example: reverse of ++**

**Lemma** \(\text{reverse}(xs ++ ys) = \text{reverse} ys ++ \text{reverse} xs\)

**Proof** by structural induction on \(xs\)

Base case:
To show: \(\text{reverse} ([] ++ ys) = \text{reverse} ys ++ \text{reverse} []\)

\(\text{reverse} ([] ++ ys) = \text{reverse} ys\) \hspace{1cm} -- by def of ++
\n\(\text{reverse} ys ++ \text{reverse} []\) \hspace{1cm} -- by def of reverse
\n\(= \text{reverse} ys\) \hspace{1cm} -- by Lemma app.Nil2

**Lemma app.Nil2**: \(xs ++ [] = xs\)

**Proof** exercise
Induction step:
To show: reverse((x:xs)++ys) = reverse ys ++ reverse(x:xs)

reverse((x:xs) ++ ys)
= reverse(x : (xs ++ ys))  -- by def of ++
= reverse(xs ++ ys) ++ [x]  -- by def of reverse
Induction step:
To show: \( \text{reverse}((x:x) ++ ys) = \text{reverse} ys ++ \text{reverse}(x:xs) \)
\[
\begin{align*}
\text{reverse}((x:x) ++ ys) & = \text{reverse}(x : (xs ++ ys)) \quad \text{-- by def of ++} \\
& = \text{reverse}(xs ++ ys) ++ [x] \quad \text{-- by def of reverse} \\
& = (\text{reverse} ys ++ \text{reverse} xs) ++ [x] \quad \text{-- by IH}
\end{align*}
\]
\[
\begin{align*}
\text{reverse} ys ++ \text{reverse}(x:xs) & = \text{reverse} ys ++ (\text{reverse} xs ++ [x]) \quad \text{-- by def of reverse}
\end{align*}
\]

Proof heuristic
- Try QuickCheck

Proof heuristic
- Try QuickCheck
- Try to evaluate both sides to common term
- Try QuickCheck
- Try to evaluate both sides to common term
- Try induction
  - Base case: reduce both sides to a common term using function defs and lemmas
  - Induction step: reduce both sides to a common term using function defs, IH and lemmas
- If base case or induction step fails: conjecture, prove and use new lemmas
**Example: reverse of ++**

**Lemma** $\text{reverse}(xs \ ++ \ ys) = \text{reverse} \ ys \ ++ \ \text{reverse} \ xs$

**Proof** by structural induction on $xs$

**Base case:**
To show: $\text{reverse} \ ([] \ ++ \ ys) = \text{reverse} \ ys \ ++ \ \text{reverse} \ []$
- $\text{reverse} \ ([] \ ++ \ ys)$
- $= \text{reverse} \ ys \quad \text{-- by def of ++}$
- $\text{reverse} \ ys \ ++ \ \text{reverse} \ []$
- $= \text{reverse} \ ys \ ++ \ [] \quad \text{-- by def of reverse}$
- $= \text{reverse} \ ys \quad \text{-- by Lemma \ app.Nil2}$

**Lemma** $\text{app.\ Nil2:} \ xs \ ++ \ [] = xs$

---

**Proof heuristic**

- Try QuickCheck
- Try to evaluate both sides to common term
- Try induction
  - Base case: reduce both sides to a common term using function defs and lemmas
  - Induction step: reduce both sides to a common term using function defs, IH and lemmas
- If base case or induction step fails:
  conjecture, prove and use new lemmas