Did I get it right?
Part 3: Induction for Lists

Speaker: David Walker
COS 326
Princeton University

http://~cos326/notes/evaluation.php
http://~cos326/notes/reasoning.php
Last Time, we saw some proofs can be done by induction over natural numbers

It turns out the structure of natural numbers is similar in many ways to the structure of lists.

In this lecture, we'll take a look at how to do a similarly structured proofs over lists.
A Couple of Useful Functions

let rec length xs =
    match xs with
    | [] -> 0
    | x::xs -> 1 + length xs

let rec cat xs1 xs2 =
    match xs1 with
    | [] -> xs2
    | hd::tl -> hd :: cat tl xs2
Proofs About Lists

Theorem: For all lists \(xs\) and \(ys\),
\[
\text{length(cat } xs \text{ ys) = length } xs + \text{ length } ys
\]

Proof strategy:

• Proof by induction on the list \(xs\)
  – recall, a list may be of these two things:
    • \([\]\) (the empty list)
    • \(\text{hd}::\text{tl}\) (a non-empty list, where \(\text{tl}\) is shorter)
  – a proof must cover both cases: \([\]\) and \(\text{hd} :: \text{tl}\)
  – in the second case, you will often use the inductive hypothesis on the smaller list \(\text{tl}\)
  – otherwise as before:
    • use folding/eval of OCaml definitions
    • use your knowledge of OCaml evaluation
    • use lemmas/properties you know of basic operations like :: and +
Proofs About Lists

**Theorem:** For all lists \( xs \) and \( ys \),

\[
\text{length(cat } xs \text{ } ys) = \text{length } xs + \text{length } ys
\]

**Proof:** By induction on \( xs \).

\[
\text{case } xs = [ ]:
\]

\[
\text{let rec length } xs = \\
\quad \text{match } xs \text{ with} \\
\quad | [] \rightarrow 0 \\
\quad | x::xs \rightarrow 1 + \text{length } xs
\]

\[
\text{let rec cat } xs1 \text{ } xs2 = \\
\quad \text{match } xs1 \text{ with} \\
\quad | [] \rightarrow xs2 \\
\quad | \text{hd}::\text{tl} \rightarrow \text{hd} :: \text{cat } \text{tl} \text{ } xs2
\]
Proofs About Lists

Theorem: For all lists $xs$ and $ys$,
\[ \text{length}(\text{cat } xs \ ys) = \text{length } xs + \text{length } ys \]

Proof: By induction on $xs$.

case $xs = [ ]$:
\[ \text{length } (\text{cat } [ ] ys) \quad \text{(LHS of theorem)} \]
Proofs About Lists

Theorem: For all lists $xs$ and $ys$,

$$\text{length}(\text{cat } xs \ ys) = \text{length } xs + \text{length } ys$$

Proof: By induction on $xs$.

case $xs = []$:

$$\text{length } (\text{cat } [] \ ys) \quad \text{(LHS of theorem)}$$
$$= \text{length } ys \quad \text{(evaluate cat)}$$
Proofs About Lists

Theorem: For all lists \( xs \) and \( ys \),

\[
\text{length}(\text{cat } xs \ ys) = \text{length } xs + \text{length } ys
\]

Proof: By induction on \( xs \).

case \( xs = [ ] \):

\[
\begin{align*}
\text{length } (\text{cat } [ ] \ ys) & \quad \text{(LHS of theorem)} \\
= \text{length } ys & \quad \text{(evaluate cat)} \\
= 0 + (\text{length } ys) & \quad \text{(arithmetic)}
\end{align*}
\]

let rec length \( xs \) =
   match \( xs \) with
   | [] -> 0
   | x::xs -> 1 + length xs

let rec cat \( xs1 \) \( xs2 \) =
   match \( xs1 \) with
   | [] -> \( xs2 \)
   | hd::tl -> hd :: cat tl \( xs2 \)
Theorem: For all lists \( xs \) and \( ys \),
\[
\text{length}(\text{cat } xs \ \text{ys}) = \text{length } xs + \text{length } ys
\]

Proof: By induction on \( xs \).

\[
\begin{align*}
\text{case } xs = []: \\
\quad \text{length } (\text{cat } []) \ ys \\
&= \text{length } ys \quad \text{(LHS of theorem)} \\
&= 0 + (\text{length } ys) \quad \text{(evaluate cat)} \\
&= (\text{length } []) + (\text{length } ys) \quad \text{(arithmetic)} \\
&= (\text{eval length}) \\
\end{align*}
\]

case done!
Proofs About Lists

**Theorem:** For all lists $xs$ and $ys$,
\[ \text{length(cat} \; xs \; ys) = \text{length} \; xs + \text{length} \; ys \]

**Proof:** By induction on $xs$.

\[
\text{case} \; xs = \text{hd} :: \text{tl}
\]

\[
\begin{align*}
\text{let rec length} \; xs &= \text{match} \; xs \; \text{with} \\
& | \; [] \rightarrow 0 \\
& | \; x :: xs \rightarrow 1 + \text{length} \; xs \\
\end{align*}
\]

\[
\begin{align*}
\text{let rec cat} \; xs1 \; xs2 &= \text{match} \; xs1 \; \text{with} \\
& | \; [] \rightarrow xs2 \\
& | \; \text{hd} :: \text{tl} \rightarrow \text{hd} :: \text{cat} \; \text{tl} \; xs2 \\
\end{align*}
\]
Proofs About Lists

**Theorem:** For all lists \(xs\) and \(ys\),

\[
\text{length}(\text{cat} \ xs \ ys) = \text{length} \ xs + \text{length} \ ys
\]

**Proof:** By induction on \(xs\).

\[\text{case } xs = \text{hd} :: \text{tl} \]

IH: \(\text{length} (\text{cat} \ \text{tl} \ ys) = \text{length} \ \text{tl} + \text{length} \ ys\)

---

```latex
let rec length xs =
match xs with
| [] -> 0
| x::xs -> 1 + length xs
```

```latex
let rec cat xs1 xs2 =
match xs1 with
| [] -> xs2
| hd::tl -> hd :: cat tl xs2
```
Proofs About Lists

**Theorem:** For all lists \( xs \) and \( ys \),

\[
\text{length}(\text{cat} \; xs \; ys) = \text{length} \; xs + \text{length} \; ys
\]

**Proof:** By induction on \( xs \).

case \( xs = \text{hd}::\text{tl} \)

IH: \( \text{length}(\text{cat} \; \text{tl} \; ys) = \text{length} \; \text{tl} + \text{length} \; ys \)

\[
\text{length}(\text{cat}(\text{hd}::\text{tl}) \; ys) \quad \text{(LHS of theorem)}
\]

==

let rec length xs =
  match xs with
  | [] -> 0
  | x::xs -> 1 + length xs

let rec cat xs1 xs2 =
  match xs1 with
  | [] -> xs2
  | hd::tl -> hd :: cat tl xs2
Proofs About Lists

**Theorem:** For all lists \( xs \) and \( ys \),

\[ \text{length}(\text{cat } xs \ ys) = \text{length } xs + \text{length } ys \]

**Proof:** By induction on \( xs \).

\[ \text{case } xs = \text{hd} :: \text{tl} \]

IH: \( \text{length } (\text{cat } \text{tl} \ ys) = \text{length } \text{tl} + \text{length } ys \)

\[
\begin{align*}
\text{length } (\text{cat } (\text{hd} :: \text{tl}) \ ys) & \quad \text{(LHS of theorem)} \\
== \text{length } (\text{hd} :: (\text{cat } \text{tl} \ ys)) & \quad \text{(evaluate cat, take } 2^\text{nd} \text{ branch)} \\
== \\
\end{align*}
\]

\[
\begin{aligned}
\text{let rec length } xs &= \\
\text{match } xs \text{ with} & \\
| [] & \rightarrow 0 \\
| x :: xs & \rightarrow 1 + \text{length } xs \\
\end{aligned}
\]

\[
\begin{aligned}
\text{let rec cat } xs1 \ xs2 &= \\
\text{match } xs1 \text{ with} & \\
| [] & \rightarrow xs2 \\
| \text{hd} :: \text{tl} & \rightarrow \text{hd} :: (\text{cat } \text{tl} \ xs2) \\
\end{aligned}
\]
Proofs About Lists

**Theorem:** For all lists $xs$ and $ys$,

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

**Proof:** By induction on $xs$.

case $xs = \text{hd} :: \text{tl}$

IH: $\text{length} \; (\text{cat} \; \text{tl} \; ys) = \text{length} \; \text{tl} + \text{length} \; ys$

\[
\begin{align*}
\text{length} \; (\text{cat} \; (\text{hd} :: \text{tl}) \; ys) &= (\text{LHS of theorem}) \\
== \text{length} \; (\text{hd} :: (\text{cat} \; \text{tl} \; ys)) &= (\text{evaluate cat, take 2}\text{nd branch}) \\
== 1 + \text{length} \; (\text{cat} \; \text{tl} \; ys) &= (\text{evaluate length, take 2}\text{nd branch}) \\
== &
\end{align*}
\]

let rec length xs =
match xs with
| [] -> 0
| x::xs -> 1 + length xs

let rec cat xs1 xs2 =
match xs1 with
| [] -> xs2
| hd::tl -> hd :: cat tl xs2
Proofs About Lists

Theorem: For all lists xs and ys,
\[ \text{length}(\text{cat } xs \ ys) = \text{length } xs + \text{length } ys \]

Proof: By induction on xs.

\[
\text{case } xs = \text{hd}::\text{tl} \\
\quad \text{IH: length (cat } \text{tl } ys) = \text{length } \text{tl} + \text{length } ys
\]

\[
\begin{align*}
\text{length (cat (hd::tl) } ys) & \quad \text{(LHS of theorem)} \\
== \text{length (hd :: (cat } \text{tl } ys)) & \quad \text{(evaluate cat, take 2\text{nd branch})} \\
== 1 + \text{length (cat } \text{tl } ys) & \quad \text{(evaluate length, take 2\text{nd branch})} \\
== 1 + (\text{length } \text{tl} + \text{length } ys) & \quad \text{(by IH)} \\
== \\
\end{align*}
\]

\[
\text{let rec length } xs = \\
\quad \text{match } xs \text{ with} \\
\quad | [] -> 0 \\
\quad | x::xs -> 1 + \text{length } xs
\]

\[
\text{let rec cat } xs1 \ xs2 = \\
\quad \text{match } xs1 \text{ with} \\
\quad | [] -> xs2 \\
\quad | \text{hd}::\text{tl} -> \text{hd} :: \text{cat } \text{tl } xs2
\]
Proofs About Lists

**Theorem:** For all lists $xs$ and $ys$,
\[
\text{length(} \text{cat } xs \text{ } ys) = \text{length } xs + \text{length } ys
\]

**Proof:** By induction on $xs$.

**case** $xs = \text{hd}::\text{tl}$

IH: $\text{length (} \text{cat } \text{tl} \text{ } ys) = \text{length } \text{tl} + \text{length } ys$

\[
\begin{align*}
\text{length (} \text{cat } (\text{hd}::\text{tl}) \text{ } ys) & \quad \text{(LHS of theorem)} \\
\quad & = \text{length } (\text{hd} :: (\text{cat } \text{tl} \text{ } ys)) \quad \text{(evaluate cat, take 2\textsuperscript{nd} branch)} \\
\quad & = 1 + \text{length (} \text{cat } \text{tl} \text{ } ys) \quad \text{(evaluate length, take 2\textsuperscript{nd} branch)} \\
\quad & = 1 + (\text{length } \text{tl} + \text{length } ys) \quad \text{(by IH)} \\
\quad & = \text{length (} \text{hd}::\text{tl}) + \text{length } ys \quad \text{(reparenthesizing and evaling length in reverse}} \\
\quad & \text{we have RHS with } \text{hd}::\text{tl} \text{ for } xs
\end{align*}
\]

case done!

\[
\text{let rec length } xs = \\
\text{match } xs \text{ with} \\
| [] -> 0 \\
| x::xs -> 1 + \text{length } xs
\]

\[
\text{let rec cat } xs1 \text{ } xs2 = \\
\text{match } xs1 \text{ with} \\
| [] -> xs2 \\
| \text{hd}::\text{tl} -> \text{hd} :: \text{cat } \text{tl} \text{ } xs2
\]
Proofs About Lists

Theorem: For all lists $xs$ and $ys$,

$$\text{length}(\text{cat } xs\ ys) = \text{length } xs + \text{length } ys$$

Proof strategy:

- **Proof by induction on the list $xs$? why not on the list $ys$?**
  - answering that question, may be the hardest part of the proof!
  - it tells you how to split up your cases
  - sometimes you just need to do some trial and error

```ocaml
let rec cat xs1 xs2 =
  match xs1 with
  | [] -> xs2
  | x::xs -> hd::tl -> hd :: cat tl xs2
```

```ocaml
let rec length xs =
  match xs with
  | [] -> 0
  | x::xs -> 1 + length xs
```

a clue: pattern matching on first argument. In the theorem: $\text{cat } xs\ ys$

Hence induction on $xs$. Case split the same as the program
Theorem: For all lists \( xs \) and \( ys \),
\[
\text{length(cat } xs \text{ ys)} = \text{length } xs + \text{length } ys
\]
Proof: By induction on \( xs \).

\[
\text{case } xs = \text{hd}::\text{tl}
\]
IH: \( \text{length (cat tl ys)} = \text{length } tl + \text{length } ys \)

\[
\begin{align*}
\text{length (cat (hd::tl) ys)} &= \text{length (hd :: (cat tl ys))} \\
&= 1 + \text{length (cat tl ys)} \\
&= 1 + (\text{length } tl + \text{length } ys) \quad \text{(by IH)} \\
&= \text{length (hd::tl) + length } ys \\
&= \text{length (hd::tl) + length } ys \\
&\quad \text{(reparenthesizing and evaling length in reverse)}
\end{align*}
\]

In your proofs, it should be really obvious

• which variable the IH is supposed to be a function of
• that your induction is on that variable
• that you’re applying the IH at smaller values

If you’re not sure it’s obvious, just say explicitly in your proof: which variable it is, and why you claim you’re applying it at smaller values
Be careful with the Induction Hypothesis!

Theorem: For all lists xs and ys,

\[ \text{length(cat} \; \text{x}s \; \text{y}s) = \text{length} \; \text{x}s + \text{length} \; \text{y}s \]

Proof: By induction on \( \text{x}s \).

In more complicated proofs, the induction hypothesis is a function of one structure where the ordering of elements in the structure is well-founded (there are no infinite descending chains).

EG: Induction on pairs of naturals (x, y) where pairs are ordered lexicographically:

\( (x_1, y_1) > (x_2, y_2) \)
iff
\( x_1 > x_2 \) or \( (x_1 = x_2 \text{ and } y_1 > y_2) \)

In COS 326, the induction hypothesis will typically be a function of one variable (in this case, \( \text{x}s \)).
Theorem: For all lists $xs$,

$$\text{add\_all} \ (\text{add\_all} \ xs \ a) \ b == \text{add\_all} \ xs \ (a+b)$$
Another List example

**Theorem:** For all lists \( xs \),

\[
\text{add\_all (add\_all xs a) b == add\_all xs (a+b)}
\]

**Proof:** By induction on \( xs \).

```ocaml
let rec add_all xs c =
  match xs with
  | [] -> []
  | hd::tl -> (hd+c)::add_all tl c
```
Another List example

Theorem: For all lists xs,

\[ \text{add\_all} \left( \text{add\_all} \, \text{xs} \, a \right) \, b = \text{add\_all} \, \text{xs} \, (a+b) \]

Proof: By induction on xs.

```
case \text{xs} = [ ]:

\text{add\_all} \left( \text{add\_all} \, [ ] \, a \right) \, b \quad (\text{LHS of theorem})
```

```
Another List example

**Theorem:** For all lists $xs$,

$$\text{add\_all}\ (\text{add\_all}\ xs\ a)\ b\ ==\ \text{add\_all}\ xs\ (a+b)$$

**Proof:** By induction on $xs$.

**case $xs = \[]:**

- $\text{add\_all}\ (\text{add\_all}\ \[]\ a)\ b\ \ (\text{LHS of theorem})$
- $\text{add\_all}\ \[]\ b\ \ (\text{by evaluation of add\_all})$
- $\text{add\_all}\ \[]\ b$

let rec add_all xs c =
    match xs with
    | \[] -> \[]
    | hd::tl -> (hd+c)::add_all tl c
Theorem: For all lists \(xs\),
\[
\text{add\_all} \ (\text{add\_all} \ xs \ a) \ b \ == \ \text{add\_all} \ xs \ (a+b)
\]

Proof: By induction on \(xs\).

```
case \(xs = \ []\):

    \text{add\_all} \ (\text{add\_all} \ [] \ a) \ b \quad \text{(LHS of theorem)}
= \text{add\_all} \ [] \ b \quad \text{(by evaluation of \text{add\_all})}
= [ ] \quad \text{(by evaluation of \text{add\_all})}
= 
```

let rec add_all xs c =
  match xs with
  | [] -> []
  | hd::tl -> (hd+c)::add_all tl c
Theorem: For all lists \(xs\),
\[
\text{add\_all (add\_all \,xs\, a) b} \,==\, \text{add\_all \,xs\,(a+b)}
\]

Proof: By induction on \(xs\).

\[
\text{case } xs = [ ]:
\]

\[
\begin{align*}
\text{add\_all (add\_all \,[\,] a) b} \, & \quad \text{(LHS of theorem)} \\
== \text{add\_all \,[\,] b} \, & \quad \text{(by evaluation of add\_all)} \\
== \,[\,] \, & \quad \text{(by evaluation of add\_all)} \\
== \text{add\_all \,[\,] (a + b)} \, & \quad \text{(by evaluation of add\_all)}
\end{align*}
\]
Theorem: For all lists $xs$,

$$\text{add}_{\text{all}}(\text{add}_{\text{all}}xs\ a)\ b\ =\ =\ \text{add}_{\text{all}}xs\ (a+b)$$

Proof: By induction on $xs$.

case $xs = \text{hd}::\text{tl}$:

$$\text{add}_{\text{all}}(\text{add}_{\text{all}}(\text{hd}::\text{tl})\ a)\ b \quad \text{(LHS of theorem)}$$

==
Another List example

**Theorem:** For all lists $xs$,

$$\text{add\_all}\ (\text{add\_all}\ xs\ a)\ b\ ==\ \text{add\_all}\ xs\ (a+b)$$

**Proof:** By induction on $xs$.

$\text{case}\ xs = \text{hd}::\text{tl}$:

$$\text{add\_all}\ (\text{add\_all}\ (\text{hd}::\text{tl})\ a)\ b\quad (\text{LHS of theorem})$$
$$==\ \text{add\_all}\ ((\text{hd}+a)\ ::\ \text{add\_all}\ \text{tl}\ a)\ b\quad (\text{by eval inner add\_all})$$
$$==$$

let rec add_all xs c =  
match xs with  
| [] -> []  
| hd::tl -> (hd+c)::add_all tl c
Another List example

Theorem: For all lists \( xs \),
\[
\text{add\_all} \ (\text{add\_all} \ xs \ a) \ b \ =\ = \ \text{add\_all} \ xs \ (a+b)
\]

Proof: By induction on \( xs \).

case \( xs = \text{hd} :: \text{tl} \):

\[
\begin{align*}
\text{add\_all} \ (\text{add\_all} \ (\text{hd} :: \text{tl}) \ a) \ b & \quad \text{(LHS of theorem)} \\
= \text{add\_all} \ ((\text{hd}+a) :: \text{add\_all} \ \text{tl} \ a) \ b & \quad \text{(by eval inner add\_all)} \\
= (\text{hd}+a+b) :: (\text{add\_all} \ (\text{add\_all} \ \text{tl} \ a) \ b) & \quad \text{(by eval outer add\_all)} \\
= & \\
\end{align*}
\]
Another List example

Theorem: For all lists $xs$, 

$$\text{add\_all} \ (\text{add\_all} \ xs \ a) \ b \ =\ = \ \text{add\_all} \ xs \ (a+b)$$

Proof: By induction on $xs$.

\[
\text{case } xs = \text{hd} :: \text{tl}:
\]

\[
\begin{align*}
\text{add\_all} \ (\text{add\_all} \ (\text{hd} :: \text{tl}) a) b & \quad \text{(LHS of theorem)} \\
= \text{add\_all} \ ((\text{hd}+a) :: \text{add\_all tl a}) b & \quad \text{(by eval inner add\_all)} \\
= (\text{hd}+a+b) :: (\text{add\_all (add\_all tl a) b}) & \quad \text{(by eval outer add\_all)} \\
= (\text{hd}+a+b) :: \text{add\_all tl (a+b)} & \quad \text{(by IH)}
\end{align*}
\]
Theorem: For all lists xs,

\[
\text{add\_all } (\text{add\_all } xs \ a) \ b = \text{ add\_all } xs \ (a+b)
\]

Proof: By induction on xs.

\[
\begin{align*}
\text{case } xs = \text{hd :: tl:} & \\
\text{add\_all } (\text{add\_all } (\text{hd :: tl}) \ a) \ b & \quad \text{(LHS of theorem)} \\
\quad = \text{add\_all } ((\text{hd+a}) :: \text{add\_all } \text{tl} \ a) \ b & \quad \text{(by eval inner add\_all)} \\
\quad = (\text{hd+a+b}) :: (\text{add\_all } (\text{add\_all } \text{tl} \ a) \ b) & \quad \text{(by eval outer add\_all)} \\
\quad = (\text{hd+a+b}) :: \text{add\_all } \text{tl} \ (a+b) & \quad \text{(by IH)} \\
\quad = (\text{hd+(a+b)}) :: \text{add\_all } \text{tl} \ (a+b) & \quad \text{(associativity of + )}
\end{align*}
\]

let rec add_all xs c =
match xs with
| [ ] -> [ ]
| hd::tl -> (hd+c)::add_all tl c
Another List example

Theorem: For all lists \(xs\),
\[
\text{add\_all}\ (\text{add\_all}\ xs\ a)\ b\ ==\ \text{add\_all}\ xs\ (a+b)
\]

Proof: By induction on \(xs\).

case \(xs = \text{hd :: tl}\):

\[
\begin{align*}
\text{add\_all}\ (\text{add\_all}\ (\text{hd :: tl})\ a)\ b & \quad \text{(LHS of theorem)} \\
==\ \text{add\_all}\ ((\text{hd}+a) :: \text{add\_all}\ \text{tl}\ a)\ b & \quad \text{(by eval inner add\_all)} \\
==\ (\text{hd}+a+b) :: (\text{add\_all}\ (\text{add\_all}\ \text{tl}\ a)\ b) & \quad \text{(by eval outer add\_all)} \\
==\ (\text{hd}+a+b) :: \text{add\_all}\ \text{tl}\ (a+b) & \quad \text{(by IH)} \\
==\ (\text{hd}+(a+b)) :: \text{add\_all}\ \text{tl}\ (a+b) & \quad \text{(associativity of +)} \\
==\ \text{add\_all}\ (\text{hd}::\text{tl})\ (a+b) & \quad \text{(by (reverse) eval of add\_all)}
\end{align*}
\]

let rec add_all xs c =
    match xs with
    | [] -> []
    | hd::tl -> (hd+c)::add_all tl c
Template for Inductive Proofs on Lists

**Theorem:** For all lists \( xs \), property of \( xs \).

**Proof:** By induction on lists \( xs \).

Case: \( xs == [ ] \):
...

Case: \( xs == \mathsf{hd} :: \mathsf{tl} \):
...

There are other ways to cover all lists:
case for \([\]\), case for \( x1::[] \), case for \( x1::x2::\mathsf{tl}' \)

But that's the same as covering \([\]\) and \( x1::\mathsf{tl} \) ...

... and then just splitting \( x1::\mathsf{tl} \) into 2 additional cases
where \( \mathsf{tl} \) is \([\]\) or \( \mathsf{tl} \) is \( x2::\mathsf{tl}' \) ...
Template for Inductive Proofs on *any datatype*

type ty = A of ... | B of ... | C of ... | D

**Theorem:** For all ty \( x \), property of \( x \).

**Proof:** By induction on \( x \) of type ty.

Case: \( x == A(...) \):
    ...
Case: \( x == B(...) \):
    ...
Case: \( x == C(...) \):
    ...
Case: \( x == D \):
    ...

cases must cover all the constructors of the datatype
SUMMARY
Summary of Proof Techniques

Proofs about programs are structured similarly to the programs:
- types tell you the kinds of values your proofs/programs operate over
- types suggest how to break down proofs/programs into cases
- when programs use recursion on smaller values they terminate and their proofs appeal to the inductive hypothesis on smaller values

Key proof ideas:
- expression evaluation: if e evaluates to e' then e == e
- substitution of equals for equals
- use well-established axioms about primitives (+, -, %, etc)
- use proof by induction to prove correctness of recursive functions
- split proofs about complex data into cases; be sure to cover all cases