Today

List induction

1. State what you’re trying to prove!
2. State and prove the base case (often empty list)
3. Assume it’s true for sublists — inductive hypothesis
4. Show that it holds for the full list
List fact

\[ \text{len (map f lst)} = \text{len lst} \]

What does this say?
Does it make sense?

List induction

Base case: \( \text{lst} = [] \)
Want to prove: \( \text{len (map f [])} = \text{len []} \)

Proof?

Prove: \( \text{len (map f lst)} = \text{len lst} \)

\begin{verbatim}
fun len [] = 0
  | len (x::xs) = 1 + len xs
fun map f [] = []
  | map f (x::xs) = (f x) :: (map f xs);
\end{verbatim}

Facts
Prove: \( \text{len} \ (\text{map} \ f \ \text{lst}) = \text{len} \ \text{lst} \)

Inductive hypothesis: \( \text{len} \ (\text{map} \ f \ \text{xs}) = \text{len} \ \text{xs} \)

Want to prove: \( \text{len} \ (\text{map} \ f \ (\text{x}::\text{xs})) = \text{len} \ (\text{x}::\text{xs}) \)

Proof?

\[
\begin{align*}
\text{fun} & \ \text{len} \ [] = 0 \\
& \ \text{len} \ \text{(x}::\text{xs}) = 1 + \text{len} \ \text{xs}
\end{align*}
\]

\[
\begin{align*}
\text{fun} & \ \text{map} \ f \ [] = [] \\
& \ \text{map} \ f \ \text{(x}::\text{xs}) = (f \ \text{x}) :: (\text{map} \ f \ \text{xs})
\end{align*}
\]

Some list “facts”

1. \( \text{len} [] = v \)
2. \( \text{len} [\text{u}] = u \)
3. \( \text{len} \ [(\text{v} :: \text{w})] = \text{len} \ [(\text{v} :: \text{w})] \)
4. \( \text{len} \ [\text{u} :: \text{v}] = \text{len} \ [\text{u} :: \text{v}] \)

What do they say?
Another list fact

len (xlst @ ylst) = len xlst + len ylst

What does this say?
Does it make sense?

1. State what you’re trying to prove!
2. State and prove the base case (often empty list)
3. Assume it’s true for smaller lists – inductive hypothesis
4. Show that it holds for the current list

Base case:  

xlst = []

Want to prove:  len ([] @ ylst) = len [] + len ylst

Proof?

Prove:  len (xlst @ ylst) = len xlst + len ylst

1. []@[v] = v
2. u@[v] = u
3. (u@[v])@[w] = u@[v][w]
4. []@[] = []

use induction on xlst

Base case:  

xlst = []

Want to prove:  len ([] @ ylst) = len [] + len ylst

Proof:  len ([] @ ylst) = ... = len [] + len ylist

1. start with left hand side
2. show a set of justified steps that derive the right hand side

Prove:  len (xlst @ ylst) = len xlst + len ylst

1. []@[v] = v
2. u@[v] = u
3. (u@[v])@[w] = u@[v][w]
4. []@[] = []
Prove:
\[ \text{len}(\text{xlst} @ \text{ylst}) = \text{len}(\text{xlst}) + \text{len}(\text{ylst}) \]

**Base case:** \( \text{xlst} = [] \)

Want to prove:
\[ \text{len}([] @ \text{ylst}) = \text{len}([]) + \text{len}(\text{ylst}) \]

\[
\begin{align*}
\text{len}([] @ \text{ylst}) &= \text{len}(\text{ylst}) & \text{fact 1} \\
&= 0 + \text{len}(\text{ylst}) & \text{math} \\
&= \text{len}([]) + \text{len}(\text{ylst}) & \text{definition of len}
\end{align*}
\]

Prove:
\[ \text{len}(\text{xlst} @ \text{ylst}) = \text{len}(\text{xlst}) + \text{len}(\text{ylst}) \]

**Inductive hypothesis:** \( \text{len}(\text{xs} @ \text{ylst}) = \text{len}(\text{xs}) + \text{len}(\text{ylst}) \)

Want to prove:
\[ \text{len}((\text{x}::\text{xs}) @ \text{ylst}) = \text{len}((\text{x}::\text{xs})) + \text{len}(\text{ylst}) \]

\[
\begin{align*}
\text{len}((\text{x}::\text{xs}) @ \text{ylst}) &= \text{len}(\text{x}::\text{xs}) + \text{len}(\text{ylst}) \\
&= \text{len}((\text{x}::\text{xs}) @ \text{ylst}) = \text{len}(\text{xs} @ \text{ylst}) = \text{len}(\text{xs}) + \text{len}(\text{ylst}) \\
&= \text{len}(\text{x}::\text{xs}) + \text{len}(\text{ylst}) & \text{definition of len}
\end{align*}
\]
Want to prove: \[ \text{len}(\langle x :: xs @ ylst \rangle) = \text{len}(x :: xs) + \text{len} ylst \]

Inductive hypothesis: \[ \text{len}(\langle x :: xs @ ylst \rangle) = \text{len}(x :: xs) + \text{len} ylst \]

Want to prove: \[ \text{len}(\langle x :: xs @ ylst \rangle) = \text{len}(x :: xs) + \text{len} ylst \]

\[
\begin{align*}
\text{len}(\langle x :: xs @ ylst \rangle) &= \text{len}(\langle x :: (xs @ ylst) \rangle) \\
&= \text{len}(\langle x :: xs @ ylst \rangle) + \text{len}(xs)
\end{align*}
\]

Blast from the past

What does the anonymous function do?

\[
\text{fun cart } \langle \rangle = \langle \rangle
\]

\[
1 \text{ cart } (u :: us) \langle v \rangle \equiv (\text{map } \lambda x \rightarrow (u, x)) \langle v \rangle @ \langle \text{cart us v} \rangle;
\]

Takes a value, \( x \), and creates a tuple with \( u \) as the first element and \( x \) as the second.
What does the map part of this function do?

For each element in vl, creates a tuple (pair) with u as the first element and an element of vl as the second.

What is the type signature?

What does this function do?

4. [2 points] Write a function `cartesian` that takes two lists and forms a list of all the ordered pairs, with one element from the first list and one from the second. For example, `cartesian [1,3,1] [2,4]` will return `[(1,2),(1,4),(3,2),(3,4),(1,2),(5,4)]`

```
cartesian : 'a list -> 'b list -> ('a * 'b) list
```
Blast from the past

Name the actor and movie

A property of cart

```haskell
fun cart :: [a] -> [b] -> [c]
  | cart (u::us) vl = (map (fn x -> (u,x)) vl) @ (cart us vl);

len(cart ul vl) = (len ul) * (len vl)
```

What does this say?
Does it make sense?

Proof by induction. Which variable, ul or vl?
Base case: \( \text{ulst} = [] \)
Want to prove: \( \text{len (cart} [\text{} \text{]} \text{vl}) = (\text{len} []) \ast (\text{len vl}) \)

Proof:

\[
\begin{align*}
\text{fun len} [] &= 0 \\
\text{len (x:xs)} &= 1 + \text{len xs}
\end{align*}
\]

Inductive hypothesis: \( \text{len (cart us vl)} = (\text{len us}) \ast (\text{len vl}) \)
Want to prove: \( \text{len (cart (us::us) vl)} = (\text{len (us::us)}) \ast (\text{len vl}) \)

Proof: \( \text{len (cart ul vl)} = (\text{len ul}) \ast (\text{len vl}) \)

\[
\begin{align*}
\text{fun len} [] &= 0 \\
\text{len (x:xs)} &= 1 + \text{len xs}
\end{align*}
\]
Want to prove: \( \text{len} (\text{cart} (\text{us}::\text{us})) \cdot \text{len} (\text{vl}) = \text{len} (\text{map} (\text{fn} \ x \Rightarrow (\text{us}, x)) \text{vl}) \cdot \text{len} (\text{cart} \text{us}) \) \\

\[
\text{IH: } \text{len} (\text{cart} \text{us}) = (\text{len} \text{us}) \cdot (\text{len} \text{vl}) \\
\text{IH: } \text{len} (\text{cart} \text{us}) = (\text{len} \text{us}) \cdot (\text{len} \text{vl})
\]

Quick refresher: datatypes

```plaintext
datatype direction = North | South | East | West;

datatype student = Firstyear of string |
| Sophomore of string |
| Junior of string |
| Senior of string |

datatype cs52int = Pos of int list |
| Zero |
| Neg of int list |
```

Recursive datatype

```plaintext
datatype 'a binTree = Empty |
| Node of 'a binTree * 'a * 'a binTree;
```

What is this?
A binary tree is a recursive data structure where each node in the tree consists of a value and then two other binary trees.

What does this look like?

Node(Empty, 1, Empty);

Node(Node(Empty, 3, Node(Empty, 4, Empty)), 5, Node(Empty, 9, Empty));
Recursive datatype

```haskell
datatype 'a binTree =
  Empty
| Node of 'a binTree * 'a * 'a binTree;
Node(Node(Empty, 3, Node(Empty, 4, Empty)), 5, Node(Empty, 9, Empty));
```

What does this look like?

Recursive datatype

```haskell
datatype 'a binTree =
  Empty
| Node of 'a binTree * 'a * 'a binTree;
Node(Node(Empty, "apple", Node(Empty, "banana", Empty)),
  "carrot",
  Node(Empty, "rhubarb", Empty));
```

Facts about binary trees

```haskell
datatype 'a binTree =
  Empty
| Node of 'a binTree * 'a * 'a binTree;
```

Counting elements in a tree $N(\cdot)$:

$N(\text{Empty}) =$

How many Nodes (i.e. values) are in an empty binary tree?
Facts about binary trees

```haskell
datatype 'a binTree =
    Empty
  | Node of 'a binTree * 'a * 'a binTree;
```

Counting elements in a tree \(N\):

- \(N(\text{Empty})\) = 0
- \(N(\text{Node}(u, elt, v))\) = 1 + \(N(u)\) + \(N(v)\)

One element stored in this node plus the nodes in the left tree and the nodes in the right tree

Leaves

A "leaf" is a Node at the bottom of the tree, i.e., \(\text{Node}(\text{Empty}, \text{elt}, \text{Empty})\)

Node(Node(Empty, 3, Node(Empty, 4, Empty)), 5, Node(Empty, 9, Empty));

\[
\begin{array}{c}
3 \\
4 \\
\end{array}
\quad
\begin{array}{c}
5 \\
9 \\
\end{array}
\]

Which are the leaves?
A "leaf" is a Node at the bottom of the tree, i.e. 
Node(Empty, elt, Empty)

Node(Node(Empty, 3, Node(Empty, 4, Empty)), 5, Node(Empty, 9, Empty));

A "leaf" is a Node at the bottom of the tree, i.e. 
Node(Empty, elt, Empty)

Node(Node(Empty, 3, Node(Empty, 4, Empty)), 5, Node(Empty, 9, Empty));

Facts about binary trees

data type 'a binTree =
| Empty
| Node of 'a binTree * 'a * 'a binTree;

Counting leaves in a tree L( ) :
L(Empty) = 0
L((Empty, elt, Empty)) = 1
L(Node(u, elt, v)) = L(u) + L(v)

Facts about binary trees

data type 'a binTree =
| Empty
| Node of 'a binTree * 'a * 'a binTree;

Counting leaves in a tree L( ) :
L(Empty) = 0
L((Empty, elt, Empty)) = 1
L(Node(u, elt, v)) = L(u) + L(v)

Facts about binary trees

data type 'a binTree =
| Empty
| Node of 'a binTree * 'a * 'a binTree;

Counting empty nodes in a tree E( ) :
E(Empty) = 0
E((Empty, elt, Empty)) = 1
E(Node(u, elt, v)) = E(u) + E(v)
Facts about binary trees

datatype 'a binTree =
Empty
| Node of 'a binTree * 'a * 'a binTree;

Counting Empty nodes in a tree E( ):

\[ E(\text{Empty}) = 1 \]
\[ E(\text{Node}(u, elt, v)) = E(u) + E(v) \]

Notation summarized

- \( N( ) \): number of elements/values in the tree
- \( L( ) \): number of leaves in the tree
- \( E( ) \): number of Empty nodes in the tree

Tree induction

1. State what you’re trying to prove!
2. State and prove the base case(s)
   (often Empty and/or Leaf)
3. Assume it’s true for smaller subtrees – inductive hypothesis
4. Show that it holds for the full tree

\[ N(\text{Empty}) = 0 \]
\[ N(\text{Node}(u, elt, v)) = 1 + N(u) + N(v) \]
\[ E(\text{Empty}) = 1 \]
\[ E(\text{Node}(u, elt, v)) = E(u) + E(v) \]
\[ L(\text{Empty}) = 0 \]
\[ L(\text{Node}(u, elt, v)) = L(u) + L(v) \]
\[ N(t) = E(t) - 1 \]

Number of nodes/values is equal to the number of Emptyys minus one

Sanity check: is it right here?

Base case: \( t = \text{Empty} \)

Want to prove: \( N(\text{Empty}) = E(\text{Empty}) - 1 \)

Proof:

\[
\begin{align*}
N(\text{Empty}) &= 0 \\
N(\text{Node}(u, elt, v)) &= 1 + N(u) + N(v) \\
E(\text{Empty}) &= 1 \\
E(\text{Node}(u, elt, v)) &= E(u) + E(v) \\
L(\text{Empty}) &= 0 \\
L(\text{Node}(u, elt, v)) &= L(u) + L(v)
\end{align*}
\]
Inductive hypotheses:

\[ N(u) = E(u) - 1 \]
\[ N(v) = E(v) - 1 \]

(Relation holds for any subtree)

Want to prove:

\[ N(\text{Node}(u, \text{elt}, v)) = E(\text{Node}(u, \text{elt}, v)) - 1 \]

Prove: \[ N(t) = E(t) - 1 \]

<table>
<thead>
<tr>
<th>N(Empty)</th>
<th>E(Empty)</th>
<th>L(Empty)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ N(\text{Node}(u, \text{elt}, v)) = 1 + N(u) + N(v) \]

Other interesting tree facts

Want to prove:

\[ N(\text{Node}(u, \text{elt}, v)) = E(\text{Node}(u, \text{elt}, v)) - 1 \]

\[ N(\text{Node}(u, \text{elt}, v)) = 1 + N(u) + N(v) \]

\[ E(\text{Node}(u, \text{elt}, v)) = E(u) + E(v) \]

\[ L(\text{Node}(u, \text{elt}, v)) = L(u) + L(v) \]
Summary of induction proofs

Numbers:
\[ \sum_{i=1}^{2^n} i = 2^i - 1 \]
\[ \sum_{i=1}^{n} \frac{(n+1)!}{2} \]

Recurrence relations:
\[ \text{count}_c(k) = \frac{k(k+1)}{2} \]
\[ \text{count}_t(k) = 2^k - k = 2 \]

Code equivalence:
\[ \text{fibrec}(n) = \text{fibiter}(n) \]

Induction on lists:
\[ \text{len} \left( \text{map} \ f \ \text{xlist} \right) = \text{len} \ \text{xlist} \]
\[ \text{len} \left( \text{xlist} @ \ \text{ylist} \right) = \text{len} \ \text{xlist} + \text{len} \ \text{ylist} \]
\[ \text{len} \left( \text{concat} \ \text{ul} \ \text{vl} \right) = \left( \text{len} \ \text{ul} \right) \times \left( \text{len} \ \text{vl} \right) \]

Induction on trees:
\[ N(t) = E(t) - 1 \]

Outline for a “good” proof by induction

1. Prove: what_to_prove

2. Base case: the_base_case(s)
   a. state what you’re trying to prove
   b. show a step by step proof
      with each step clearly justified

3. Assume: the_inductive_hypothesis

4. Show: what_you’re_trying_to_prove
   step by step proof from left hand side deriving the right hand side with each step clearly justified