Structural Induction wit

Dr. Liam O'Connor
University of Edinburgh LFCS
UNSW, Term 3 2020

Recap: Induction

Definition

Let $P(x)$ be a predicate on natural numbers $x \in \mathbb{N}$. To show $\forall x \in \mathbb{N}.\ P(x)$, we can use induction:

Recap: Induction

Definition

Let $P(x)$ be a predicate on natural numbers $x \in \mathbb{N}$. To show $\forall x \in \mathbb{N}. P(x)$, we can use induction:

- Show $P(0$

Recap: Induction

Definition

Let $P(x)$ be a predicate on natural numbers $x \in \mathbb{N}$. To show $\forall x \in \mathbb{N}.\ P(x)$, we can use induction:

- Show $P(0$
- Assuming

Recap: Induction

Definition

Let $P(x)$ be a predicate on natural numbers $x \in \mathbb{N}$. To show $\forall x \in \mathbb{N}. P(x)$, we can use induction:

- Show $P(0$
- Assuming

Example (Sum of Integers)

Write a recursive function *sumTo* to sum up all intege n.

Recap: Induction

Definition

Let $P(x)$ be a predicate on natural numbers $x \in \mathbb{N}$. To show $\forall x \in \mathbb{N}.\ P(x)$, we can use induction:

- Show $P(0$
- Assuming

Example (Sum of Integers)

Write a recursive function *sumTo* to sum up all integers n. Show that:

$$\forall n \in \mathbb{N}.\ sumTo\ n = \frac{n(n+1)}{2}$$

Haskell Data Types

We can define natural numbers as a Haskell data type reflecting this inductive structure.

Haskell Data Types

We can define natural numbers as a Haskell data type reflecting this inductive structure.

Example

Define addition, prove that $\forall n.\ n + \mathrm{Z} = n$.

Haskell Data Types

We can define natural numbers as a Haskell data type reflecting this inductive structure.

Example

Define addition, prove that $\forall n.\ n + \mathsf{Z} = n$.

Inductive Structure

Observe that the non-recursive constructors correspond to base cases and the recursive constructors correspond to inductive cases

Lists

Lists are singly-linked lists in Haskell. The empty list is written as `[]` and a list node is written as `x : xs`. The value `x` is called the head and the rest of the list `xs` is called the tail. Thus:

```
"hi!" == 'h' : 'i' : '!' : []
```

Lists

Lists are singly-linked lists in Haskell. The empty list is written as `[]` and a list node is written as `x : xs`. The value `x` is called the head and the rest of the list `xs` is called the tail. Thus:

```
"hi!" == 'h':'i':'!':[]
```


When we define recursive functions on lists, we use the last form f

Example

(Re)-define the functions *length*, *take* and *drop*.

Induction on Lists

If lists weren't already defined in the standard library, we could define them ourselves:

```
data List a = Nil | Cons a (List a)
```

Induction on Lists

If lists weren't already defined in the standard library, we could define them ourselves:

$$\text{data } List\ a = Nil \mid Cons\ a\ (List\ a)$$

Induction

If we want to prove th

$$\forall xs.\ P(xs$$

It suffices to:

Induction on Lists

If lists weren't already defined in the standard library, we could define them ourselves:

$$\text{data } List\ a = Nil \mid Cons\ a\ (List\ a)$$

Induction

If we want to prove th

$$\forall xs.\ P(xs$$

It suffices to:

1. Show $P([])$ (the base case from nil)

Induction on Lists

If lists weren't already defined in the standard library, we could define them ourselves:

$$\texttt{data}\ \mathit{List}\ a = \mathit{Nil} \mid \mathit{Cons}\ a\ (\mathit{List}\ a)$$

Induction

If we want to prove th

$$\forall xs.\ P(xs$$

It suffices to:

1. Show $P(\texttt{[]})$ (the base case from nil)
2. Assuming the inductive hypothesis $P(\texttt{xs})$, show $P(\texttt{x:xs})$ (the inductive case from cons).

Induction on Lists

Example (Tak

- Show that

Induction on Lists

Example (Tak

- Show that
- Show that

Induction on Lists

Example (Tak

- Show that
- Show that

$\Longrightarrow$ Sometimes we must generalise the proof goal.

Induction on Lists

Example (Tak

- Show that
- Show that

$\Longrightarrow$ Sometimes we must generalise the proof goal.

$\Longrightarrow$ Sometimes we must prove auxiliary lemmas.

Binary Trees

```
data Tree a = Leaf
            | Branch a (Tree a) (Tree a)
```

Binary Trees

```
data Tree a = Leaf
            | Branch a (Tree a) (Tree a)
```


Induction Prin

To prove a propert

- Prove the ba

Binary Trees


```
data Tree a = Leaf
            | Branch a (Tree a) (Tree a)
```


Induction Prin

To prove a propert

- Prove the ba
- Assuming the two *inductive hypotheses*:

Binary Trees

```
data Tree a = Leaf
            | Branch a (Tree a) (Tree a)
```


Induction Prin

To prove a propert

- Prove the ba
- Assuming the two *inductive hypotheses*:
  - $P(l)$ and
  - $P(r)$

Binary Trees

```
data Tree a = Leaf
            | Branch a (Tree a) (Tree a)
```


Induction Prin

To prove a propert

- Prove the ba
- Assuming the two *inductive hypotheses*:
  - $P(l)$ and
  - $P(r)$

We must show $P(Branch\ x\ l\ r)$.

Binary Trees

data *Tree a* = Leaf
| Branch *a* (*Tree a*) (*Tree a*)

Induction Prin

To prove a propert

- Prove the ba
- Assuming the two *inductive hypotheses*:
  - $P(l)$ and
  - $P(r)$

  We must show $P(Branch\ x\ l\ r)$.

Example (Tree functions)

Define *leaves* and *height*, and show $\forall t.\ height\ t < leaves\ t$

Rose Trees


```
data Forest a = Empty | Cons (Rose a) (Forest a)
```


Note that *For*

Rose Trees


```
data Forest a = Empty | Cons (Rose a) (Forest a)
```


Note that *For*

Example (Rose tree functions)

Define *size* and *height* and try to show

$$\forall t.\ height\ t \leq size\ t$$

Simultaneous Induction

To prove a property about two types defined mutually, we have to prove two properties *simultaneously*.

data *Forest a* = Empty | Cons (*Rose a*) (*Forest a*)

Inductive Principle

To prove a property *P*(*t*) about all *Rose* trees *t* and *P*(*ts*) about all *Forests* *ts* simultaneously:

Simultaneous Induction

To prove a property about two types defined mutually, we have to prove two properties *simultaneously*.


```
data Forest a = Empty | Cons (Rose a) (Forest a)
```


Inductive Principle

To prove a property $P(t)$ about all *Rose* trees t and $Q(ts)$ about all *Forests* ts simultaneously:

- Prove $Q(Empty)$

Simultaneous Induction

To prove a property about two types defined mutually, we have to prove two properties *simultaneously*.

data *Forest a* = Empty | Cons (*Rose a*) (*Forest a*)

Inductive Principle

To prove a property $P(t)$ about all *Rose* trees t and $Q(ts)$ about all *Forests* ts simultaneously:

- Prove $Q(Empty)$
- Assuming $P(t)$ and $Q(ts)$ (inductive hypotheses), show $Q(Cons\ t\ ts)$.

Simultaneous Induction

To prove a property about two types defined mutually, we have to prove two properties *simultaneously*.

data *Forest a* = Empty | Cons (*Rose a*) (*Forest a*)

Inductive Principle

To prove a property $P(t)$ about all *Rose* trees t and $Q(ts)$ about all *Forests* ts simultaneously:

- Prove $Q(Empty)$
- Assuming $P(t)$ and $Q(ts)$ (inductive hypotheses), show $Q(Cons\ t\ ts)$.
- Assuming $Q(ts)$ (inductive hypothesis), show $P(Node\ x\ ts)$.