

Structural Induction

COMP2600 — Formal Methods for Software Engineering

Jeremy Dawson

Australian National University
Semester 2, 2011

Natural Number Induction

This is the induction you already know.

To prove a property P for all natural numbers:

- Prove it for 0
- Prove that, if it is true for n it is true for $n + 1$.

The principle is usually expressed as a rule of inference:

$$\frac{P(0) \quad \forall n. P(n) \rightarrow P(n+1)}{\forall n. P(n)}$$

Structural Induction

What is covered in this topic.

- Induction on the natural numbers.
- Structural induction over Lists.
- Structural induction over Trees.
- The principle that the structural induction rule for a particular data type follows from its definition.

Why does it Work?

The natural numbers are an *inductively defined set*:

1. 0 is a natural number;
2. If n is a natural number, so is $n + 1$;

No object is a natural number unless justified by these clauses.

From the assumptions:

$$P(0) \quad \forall n. P(n) \rightarrow P(n+1)$$

we get a sequence of deductions:

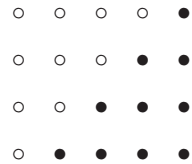
$$P(0), P(1), P(2), P(3), \dots$$

which justifies the conclusion for any n you choose.

Example of Mathematical Induction

Let's prove this property of natural numbers:

$$\sum_{i=0}^n i = \frac{n \times (n+1)}{2}$$



First the **Base case** $P(0)$

$$\sum_{i=0}^0 i = \frac{0 \times (0+1)}{2}$$

This is obviously true because both sides equal 0

Now prove $P(a+1)$, that is, $\sum_{i=0}^{a+1} i = \frac{(a+1) \cdot ((a+1)+1)}{2}$

$$\begin{aligned} \sum_{i=0}^{a+1} i &= \sum_{i=0}^a i + (a+1) \\ &= \frac{a \cdot (a+1)}{2} + (a+1) && \text{(by IH)} \\ &= \frac{a \cdot (a+1)}{2} + \frac{2 \cdot (a+1)}{2} \\ &= \frac{(a+2) \cdot (a+1)}{2} \\ &= \frac{(a+1) \cdot (a+2)}{2} \end{aligned}$$

That is, $P(a+1)$ is true

Now the **Step case** $\forall n. P(n) \rightarrow P(n+1)$

Following the \forall -I rule of natural deduction, we will first prove $P(a) \rightarrow P(a+1)$ for a particular number a , then generalise.

Assume $P(a)$. (Our proof follows the \rightarrow -I rule of natural deduction).

This assumption is called the **induction hypothesis (IH)**.

$$\sum_{i=0}^a i = \frac{a \cdot (a+1)}{2}$$

Now prove $P(a+1)$

Discharging the assumption $P(a)$, as in the \rightarrow -I rule, we have:

$$P(a) \rightarrow P(a+1)$$

Generalising over a , as in the \forall -I rule, gives

$$\forall n. P(n) \rightarrow P(n+1)$$

We have now satisfied both premises of the induction rule.

Theorem proved!

Induction on Lists

Like natural numbers, lists are also *inductively defined*.

1. $[] :: [a]$ (ie, the term $[]$ is a member of the type $[a]$)
2. If $x :: a$ and $xs :: [a]$ then $(x : xs) :: [a]$

No object is a list of a 's unless justified by these clauses.

To prove a property for all lists (whose elements have type a)

- Prove it for $[]$
- Prove that, whenever it is true for xs it is also true for $x : xs$.

That's Induction on Structure

The rule of *Structural Induction for Lists* is usually written as:

$$\frac{P([]) \quad \forall x. \forall xs. P(xs) \rightarrow P(x : xs)}{\forall xs. P(xs)}$$

or, being fussy with types:

$$\frac{P([] :: [a]) \quad \forall (x :: a). \forall (xs :: [a]). P(xs) \rightarrow P(x : xs)}{\forall (xs :: [a]). P(xs)}$$

Why does it Work?

Suppose we have proved:

- The *base case*: $P([])$
- The *step case*: $\forall x. \forall xs. P(xs) \rightarrow P(x : xs)$

We can use these facts to prove that $P([4, 2, 6])$ is true.

- $P([])$ is given
- $P([6])$ follows from $P([])$ by the inductive step (here, $xs = [], x = 6$)
- $P([2, 6])$ follows from $P([6])$ by the inductive step (here, $xs = [6], x = 2$)
- $P([4, 2, 6])$ follows from $P([2, 6])$ by the inductive step ($xs = [2, 6], x = 4$)

QED

Standard functions

Many of our examples will use some standard functions.

We will use each line of the function definition as a rewrite rule.

$$\text{length } [] = 0 \quad \text{-- (L1)}$$

$$\text{length } (x : xs) = 1 + \text{length } xs \quad \text{-- (L2)}$$

$$\text{map } f [] = [] \quad \text{-- (M1)}$$

$$\text{map } f (x : xs) = f x : \text{map } f xs \quad \text{-- (M2)}$$

$$[] ++ ys = ys \quad \text{-- (A1)}$$

$$(x : xs) ++ ys = x : (xs ++ ys) \quad \text{-- (A2)}$$

length (map f xs) = length xs

We're doing induction over the list xs , so our first step is to substitute $[]$ for xs and prove the base case.

Base Case: $P([])$

$$\text{length (map f [])} = \text{length []}$$

Now look for rewrite rules to make one side obviously equal to the other.

This holds by (M1)

What are we really doing here?

Our step case demonstrates that we can derive $P(a : as)$ from $P(as)$

1	a	as	$P(as)$	
:			\vdots	
6			$P(a : as)$	
7			$P(as) \rightarrow P(a : as)$	$\rightarrow\text{-I, 1-6}$
8			$\forall xs. P(xs) \rightarrow P(a : xs)$	$\forall\text{-I, 7}$
9			$\forall x. \forall xs. P(xs) \rightarrow P(x : xs)$	$\forall\text{-I, 8}$

Step Case: $\forall x. \forall xs. P(xs) \rightarrow P(x : xs)$

Once again, we prove this for a particular list $a : as$, then generalise.

Assume $P(as)$

$$\text{length (map f as)} = \text{length as} \quad \text{-- (IH)}$$

Prove $P(a : as)$, that is

$$\text{length (map f (a:as))} = \text{length (a:as)}$$

$$\begin{aligned}
\text{length (map f (a:as))} &= \text{length (f a : map f as)} \quad \text{-- by (M2)} \\
&= 1 + \text{length (map f as)} \quad \text{-- by (L2)} \\
&= 1 + \text{length as} \quad \text{-- by (IH)} \\
&= \text{length (a:as)} \quad \text{-- by (L2)}
\end{aligned}$$

So we have proved $P(a : as)$

We'll skip the formal generalisation step and call the Step Case proved.

length (xs ++ ys) = length xs + length ys

We do induction over one list only.

When proving the above theorem, treat one of xs or ys as a constant.

Which one? Look at how $xs ++ ys$ is defined: by recursion on xs .

So treat ys as a constant, and let $P(xs)$ be

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

Base Case: $P([])$ We want to prove

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

$$\begin{aligned}
\text{length ([] ++ ys)} &= \text{length ys} \quad \text{-- by (A1)} \\
&= 0 + \text{length ys} \\
&= \text{length []} + \text{length ys} \quad \text{-- by (L1)}
\end{aligned}$$

Step Case: $\forall x. \forall xs. P(xs) \rightarrow P(x : xs)$

Assume $P(as)$

$$\text{length } (as ++ ys) = \text{length } as + \text{length } ys \quad \text{-- (IH)}$$

Prove $P(a : as)$, that is

$$\text{length } ((a : as) ++ ys) = \text{length } (a : as) + \text{length } ys$$

$$\begin{aligned} \text{length } ((a : as) ++ ys) &= \text{length } (a : (as ++ ys)) \quad \text{-- by (A2)} \\ &= 1 + \text{length } (as ++ ys) \quad \text{-- by (L2)} \\ &= 1 + \text{length } as + \text{length } ys \quad \text{-- by (IH)} \\ &= \text{length } (a : as) + \text{length } ys \quad \text{-- by (L2)} \end{aligned}$$

Theorem proved!

$$\underline{\text{map } f \text{ (xs ++ ys) = map } f \text{ xs ++ map } f \text{ ys}}$$

Remember, induction is over one list only.

Treat ys as a constant

(why ys ? Again, the clue is the definition of $xs ++ ys$)

So let $P(xs)$ be $\text{map } f \text{ (xs ++ ys) = map } f \text{ xs ++ map } f \text{ ys}$

Base Case: $P([])$

$$\text{map } f \text{ ([] ++ ys) = map } f \text{ [] ++ map } f \text{ ys}$$

$$\begin{aligned} \text{map } f \text{ ([] ++ ys)} &= \text{map } f \text{ ys} \quad \text{-- by (A1)} \\ &= [] ++ \text{map } f \text{ ys} \quad \text{-- by (A1)} \\ &= \text{map } f \text{ [] ++ map } f \text{ ys} \quad \text{-- by (M1)} \end{aligned}$$

A few meta-points:

On the induction hypothesis:

- The *induction hypothesis* ties the recursive knot in the proof.
- If you haven't used the *induction hypothesis* the proof is probably wrong.
- It's important to know which rule the *induction hypothesis* actually is.

On rules:

- You can only use the rules you are given.
- The rules are: the function definitions, the induction hypothesis and basic arithmetic.

Step Case: $\forall x. \forall xs. P(xs) \rightarrow P(x : xs)$

Assume $P(as)$

$$\text{map } f \text{ (as ++ ys) = map } f \text{ as ++ map } f \text{ ys} \quad \text{-- (IH)}$$

Prove $P(a : as)$, that is

$$\text{map } f \text{ ((a : as) ++ ys) = map } f \text{ (a : as) ++ map } f \text{ ys}$$

$$\begin{aligned} \text{map } f \text{ ((a : as) ++ ys)} &= \text{map } f \text{ (a : (as ++ ys))} \quad \text{-- by (A2)} \\ &= f \text{ a : map } f \text{ (as ++ ys)} \quad \text{-- by (M2)} \\ &= f \text{ a : (map } f \text{ as ++ map } f \text{ ys)} \quad \text{-- by (IH)} \\ &= (f \text{ a : map } f \text{ as) ++ map } f \text{ ys} \quad \text{-- by (A2)} \\ &= \text{map } f \text{ (a : as) ++ map } f \text{ ys} \quad \text{-- by (M2)} \end{aligned}$$

Theorem proved!

Observe a Trilogy

- **Inductive Definition**

`data [a] = [] | a : [a]`

- **Recursive Function Definitions**

`f [] =`

`f (x:xs) =` (definition usually involves `f xs`)

- **Structural Induction Principle**

Prove $P([])$

Prove $\forall x. \forall xs. P(xs) \rightarrow P(x : xs)$ (proof usually uses $P(xs)$)

Each version has a base case and a step case. The form of the inductive type definition determines the form of recursive function definitions and the structural induction principle.

Finite lists only!!

- What if $P(xs)$ is “ xs is a *finite* list”?
- Try to prove $\forall xs. P(xs)$ by structural induction on lists. This works !!
- This theory works only for *finite* lists (trees, integers, etc)
- The logic of infinite structures is different