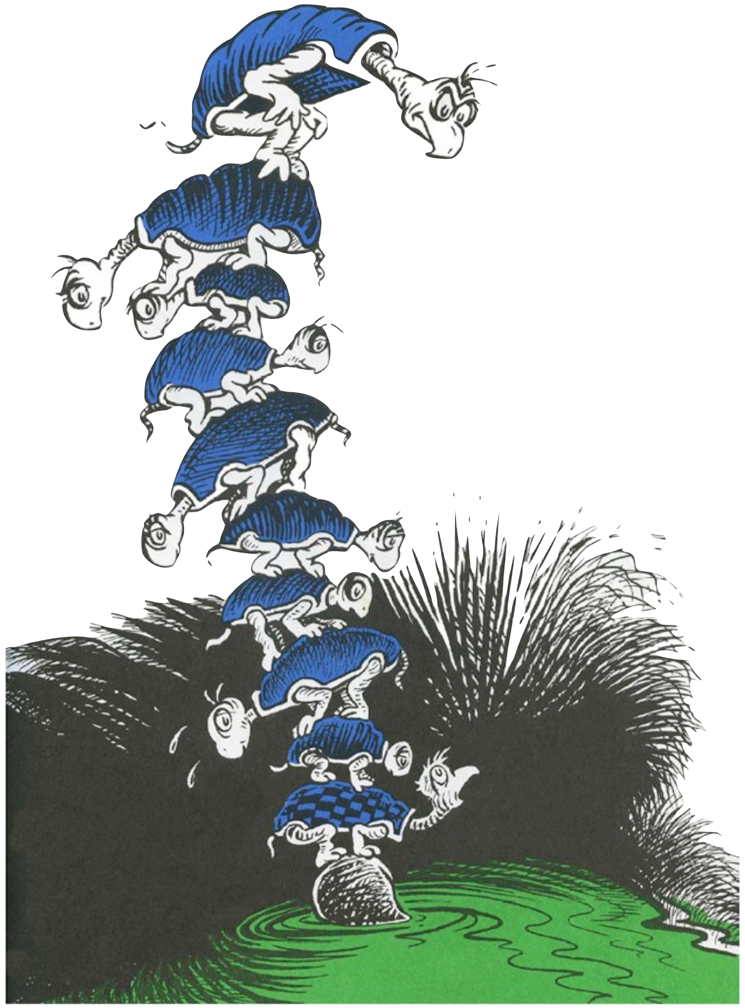
CS 565

Programming Languages (graduate)
Spring 2024

Week 2
Induction

Today



- Generate the induction principle for inductive data types

- Prove properties of inductive data types using induction.

Proof By Case Analysis

How would you justify the following claim?

b1 && (b2 && b3) = (b1 && b2) && b3

Construct a truth table that enumerates all cases:

P	Q	R	Formula
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	T
F	T	T	T
F	T	F	T
F	F	T	T
F	F	F	T


Proof By Exhaustion
TOO TIRED
MUST BE TRUE

Proof By Case Analysis

How would you justify the following fact:

> For any three numbers n, m and p,
> $n + (m + p) = (n + m) + p$.

Infinite number of cases here!

Proof By Induction

How would you justify the following fact:

> For any three numbers n, m and p,
>
> $n + (m + p) = (n + m) + p$.

Proof: By induction on n.

First, suppose $n = 0$.

We must show: $0 + (m + p) = (0 + m) + p$.

This follows directly from the definition of addition.

Induction Hypothesis

Next, suppose $n = 1 + n'$, where $n' + (m + p) = (n' + m) + p$.

We must show: $(1 + n') + (m + p) = ((1 + n') + m) + p$.

By the definition of +, this follows from $1 + (n' + (m + p)) = 1 + ((n' + m) + p)$, which is immediate from the induction hypothesis. **QED**.

Nat Induction

Mathematical Induction for Natural Numbers:

For any predicate P on natural numbers, **if:**

1. P(0)
2. P(n) implies P(n+1)

Then:

for all n, P(n) holds.

Induction


```
Inductive tree : Type :=
    | leaf
    | node (x : nat) (lt rt : tree).

Fixpoint insert  (cmp : nat -> nat -> bool)
                  (t: tree) (x : nat) : tree  :=
  match t with
  | leaf => node x leaf leaf
  | node y lt rt => if (cmp x y) then node y (insert cmp lt x) rt
                         else node y lt (insert cmp rt y)
  end.

Fixpoint element (t : tree) (n : nat) : bool :=
  match t with
  | leaf => false
  | node y lt rt =>
       orb (eqb x y) (orb (element eqb lt x) (element eqb rt x))
  end.
```

Tree Induction

Works for trees too:

> For any number n, and tree t
> element (insert t n) n = true.

Proof: By induction on t.

First, suppose t = leaf.

We must show: element (insert leaf n) n = true.

This follows directly from the definition of element.

Tree Induction

Works for trees too:

> For any number n, and tree t
> element (insert t n) n = true.

Induction Hypothesis

Proof: By induction on t.

Next, suppose t = node n' lt rt, where

element (insert lt n) n = true and element (insert rt n) n = true.

We must show: element (insert (node n' lt rt) n) n = true.

By definition, this is equivalent to:

element (**if** (cmp n n') **then** node n' (insert cmp lt n) rt
else node y lt (insert cmp rt n))

- ★ Consider the case when cmp n n′ = true.

  We must show: element (node n' (insert cmp lt n) rt) n = true.

  This follows from the IH.

- ★ Consider the case when cmp n n′ = false.

  We must show: element (node n' lt (insert cmp rt n)) n = true.

  This follows from the IH.

Tree Induction

Works for trees too:

Mathematical Induction for Binary Trees:

For any predicate Q on binary trees, **if:**

1. Q(leaf)
2. $Q(t_1)$ and $Q(t_2)$ implies Q(node n t_1 t_2)

Then:

for all t, Q(t) holds.

ADT Induction

Principle of Mathematical Induction:

For any algebraic datatype T with constructors $c_1 \ldots c_n$,

For any predicate Q on T, **if:**

1. $Q(v_1)$ and $Q(v_2)$ and … $Q(v_j)$ implies $Q(c_1\ v_1 \ldots v_j)$
2. $Q(v_1)$ and $Q(v_2)$ and … $Q(v_j)$ implies $Q(c_2\ v_1 \ldots v_j)$

   …

n. $Q(v_1)$ and $Q(v_2)$ and … $Q(v_j)$ implies $Q(cn\ v_1 \ldots v_j)$

Then:

for all t, Q(t) holds.

Lists


```
Inductive list {X : Type} : Type :=
  | nil
  | cons (x : X) (l : list).
```

Mathematical Induction for Lists:

For any predicate Q on lists, **if:**

1. Q(nil)
2. Q(l) implies Q(cons x l)

Then:

for all l, Q(l) holds.

a list constructed by adding x to the head of l

Induction on syntax trees


```
Inductive aexp : Type :=
  | ANum (a : nat)
  | APlus (a1 a2 : aexp)
  | AMinus (a1 a2 : aexp)
  | AMult  (a1 a2 : aexp).

Fixpoint aexp_opt_zero (a : aexp) : aexp :=
  match a with
   | ANum n => ANum n
   | APlus (ANum 0) e2 => aexp_opt_plus e2
   | APlus e1 e2 => APlus (aexp_opt_plus e1) (aexp_opt_plus e2)
   | AMinus e1 e2 => AMinus (aexp_opt_plus e1) (aexp_opt_plus e2)
   | AMult e1 e2 => AMult (aexp_opt_plus e1) (aexp_opt_plus e2)
  end.
```

Induction on syntax trees

- Works for abstract syntax trees too!

- Using ADT induction, we can prove in Coq:

```
Theorem aexp_opt_zero_sound
  : forall a, aeval (aexp_opt_zero a) = aeval a.
Proof.
  induction a.
  - ...
  - ...
Qed.
```